A Wireless Autonomous Vehicular Electronic Surveyor

by
Christopher Harrower
Serge LeBlanc
Julian Nedohin-Macek
Richard Rodd

Final report submitted in partial satisfaction of the requirements for the degree of

Bachelor of Science

in

Electrical and Computer Engineering

in the

Faculty of Engineering

of the

University of Manitoba

Faculty Supervisor:

Dr. Douglas Thomson

Spring 2002

Abstract

The purpose of this document is to detail the process of designing a wireless surveyor that transmits data from an electronic eye and is mounted on a vehicle that communicates with a remote computer over a Bluetooth connection.

There are several solutions for wireless surveyor systems in exploration of hazardous or remote locations using a system in which a vehicle is controlled by a stationary base system. This design project sought to investigate the feasibility of using Bluetooth wireless technology to implement a similar system in which a vehicle gathers visual data and sends it back to a remote base computer when a connection had been established. The difference between this system and other base and vehicle systems is that Bluetooth allows an impromptu search and connection with other Bluetooth devices, namely the base computer. This is useful in that many small and simple vehicles could take the place of a single larger and more complex vehicle in remote surveying. These smaller vehicles could use Bluetooth technology to interact not only with the base system, but also with each other to combine their efforts according to predetermined programming. The design illustrates a simple one vehicle, one computer system that could be the basis of a more complicated system.

Contributions

The construction of the vehicle and layout of the final Bluetooth printed circuit (PCB) board was completed by Julian Nedohin-Macek with aid from all other group members. The design of the electronic eye and layout of the first version of the Bluetooth printed circuit board (PCB) was completed by Christopher Harrower with aid from all other group members. Serge LeBlanc completed the application software for the computer-side Bluetooth module. The Microchip PIC 16F877 microcontroller software was written by Richard Rodd. Richard Rodd also drew the schematics of the Bluetooth PCB, the microcontroller-based PCB, and created the layout of the PIC microcontroller board. All group members took equal share in the writing and editing of the formal proposal, progress reports, and all drafts of this report.

Acknowledgements

We would like to thank the members of the Department of Electrical and Computer Engineering at the University of Manitoba. In particular, we appreciate the guidance of Dr. Thomson, Dr. LoVetri, Al McKay, and Ken Biegun during the course of this design project. We also wish to thank Dawn Nedohin-Macek and Lawrence Arendt of the Manitoba High Voltage Direct Current Research Center (HVDC) for their aid and suggestions. We are grateful for the donation of several components by National Semiconductor, Maxim, and Texas Instruments.

We would like to especially thank the University of Manitoba's IEEE McNaughton Centre for providing its lab space and tools throughout this project and throughout our degrees.

Table of Contents

List of Figures

List of Tables

Nomenclature

The table below lists the HCI commands and HCI events implemented in either the microcontroller code or Windows PC code for this project

Name	Hexadecimal Form	Usage
HCI_RESET	030C	Issue a reset command to the module.
HCI_READ_BUFFER_SIZE	330C	Read the size of buffers available in the module.
HCI_READ_BD_ADDR	0510	Obtain the local device's address.
HCI_SET_EVENT_FILTER		Configure masking of or automatic response to events.
HCI_WRITE_SCAN_ENABLE		Configure the acceptance or rejection of requests for availability.
HCI_CREATE_CONNECTION	0504	Attempt to create a connection with a device.
HCI_DISCONNECT		Disconnect a connection.
HCI_COMMAND_COMPLETE		Event that can also return results of the command.
HCI_COMMAND_STATUS		Event that signals a command is in progress and not complete.
HCI_CONNECTION_COMPLETE		Event indicating a new connection exists.
HCI_DISCONNECTION_COMPLETE		Event returned when a disconnect is attempted.

Glossary

Netlist Tracks how pins of various devices are electrically connected together for bringing a Capture design into Layout.

Orcad A software suite of tools distributed by Cadence that includes Capture and Layout.

Orcad Capture A software tool used to design schematics.

Orcad Layout A software tool used to design printed circuit boards.

Padstack A set of pads stacked vertically above each other on a printed circuit board.

Piconet A network of Bluetooth devices that are in active communication.

Pinout The pin arrangement of a device.

Via A physical routing feature that runs a trace from one layer to another in a printed circuit board.

Chapter 1

Introduction

1.1 Purpose

The purpose of this design project is to implement a possible method to survey an environment using a Bluetooth wireless technology equipped vehicle. The overall achievement is to deliver information over a Bluetooth communication channel. The advantage of using Bluetooth technology is to allow impromptu establishment of a connection between devices. Such a connection would be useful in implementing a system of cooperatively working vehicles.

The scope of this project is to demonstrate data transfer over the Bluetooth channel. The data in this project is optical data from the vehicle's surroundings. This data is sent to a computer for analysis.

1.2 System Overview

The system is pictured in Figure 1.1 and consists of four parts on the vehicle with a fifth part being a computer and software. The four parts of the vehicle are an electronic 'eye', a microcontroller, a Bluetooth module, and a chassis. The eye gathers optical information from the environment and transmits it to a microcontroller. The microcontroller prepares information for transmission over a Bluetooth link. The Bluetooth module and microcontroller implement the Bluetooth link. The computer receives the sent information via its own Bluetooth device. The information is then displayed and stored in a file.

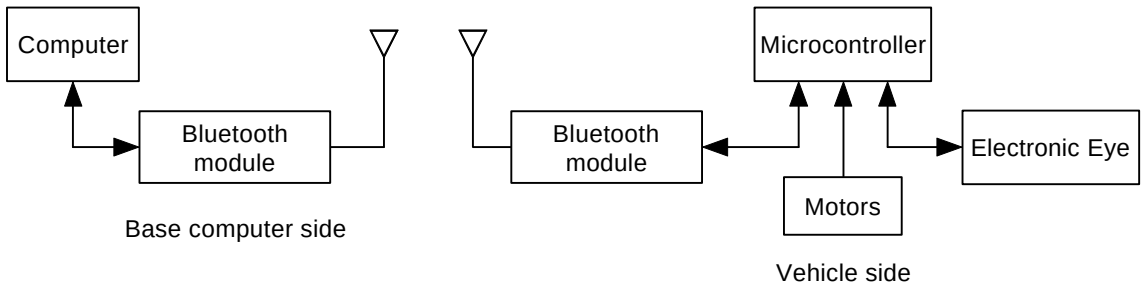


Figure 1-1. System block diagram.

Chapter 2

The Vehicle

2.1 Overview

The vehicle design consists of a chassis, motors, solar cells, and motor drive circuits. The design implements options to drive all motors from one supply, each motor from their own supply, or one motor from several supplies.

2.2 The Chassis

The chassis supports an Ericsson Bluetooth development board, the electronic eye assembly, the microcontroller, and supporting hardware.

The initial chassis of the vehicle was constructed from perforated electronics hobby board and miscellaneous parts. In the design process, many construction material options were considered. The perforated hobby board was chosen for its lightweight and ease of manipulation. The design called for a dedicated chassis to hold the Bluetooth board, the electronic eye, the microcontroller board, the motors, the drive circuits and the energy source.

Initially it was thought that the microcontroller board would be approximately 7.0 cm by 5.0 cm by 2.0 cm. Due to design considerations discussed in chapter four, the final design of the microcontroller board is 15.0 cm by 15.0 cm by 2.0 cm. The microcontroller PCB was then used as the base of the chassis instead of being mounted on a chassis.

2.3 The Drive Circuit

The drive circuit, Figure 2-1, stores the energy for the motor and controls the rate at which the storage device is discharged across the motors. Each drive circuit consists of a storage capacitor, a current limiting resistor, and triggering components. The drive circuit does not include the voltage source or motors themselves.

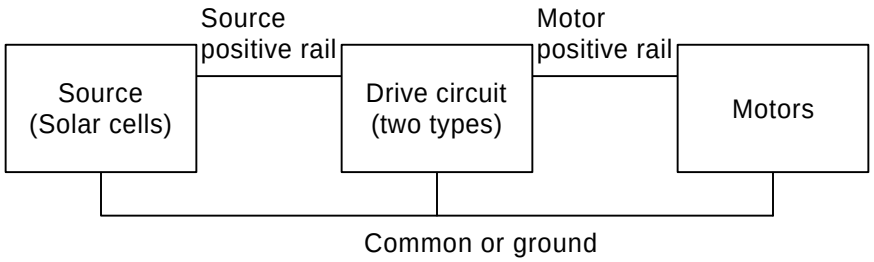


Figure 2-1. Drive circuit block diagram.

Several components were researched and qualitatively tested for their capability to trigger the drive circuit. The tested components included a voltage detector, several resistors, diodes, transistors, photoresistors, phototransistors, and combinations of these components. It was found that the most efficient of these was the voltage detector, with the diode being the second most efficient triggering device. These two components were each implemented in a drive circuit.

The first drive circuit explored is a diode-triggered motor circuit designed by Mark W. Tilden of Los Alamos National Labs for his robot projects. [7] The second drive circuit explored uses an MN1381L voltage detector integrated circuit. This MN1381L triggers when the capacitor voltage reaches approximately 3.0 V. Each of these triggering methods offers a small package and suitably efficient energy transfer from the storage capacitor to the motor.

2.3.1 Diode-triggered circuit

The diode-triggered circuit uses a single 1N4004 diode to trigger a PNP transistor into low impedance mode. Figure 2-3 offers a schematic of the diode-triggered drive circuit.

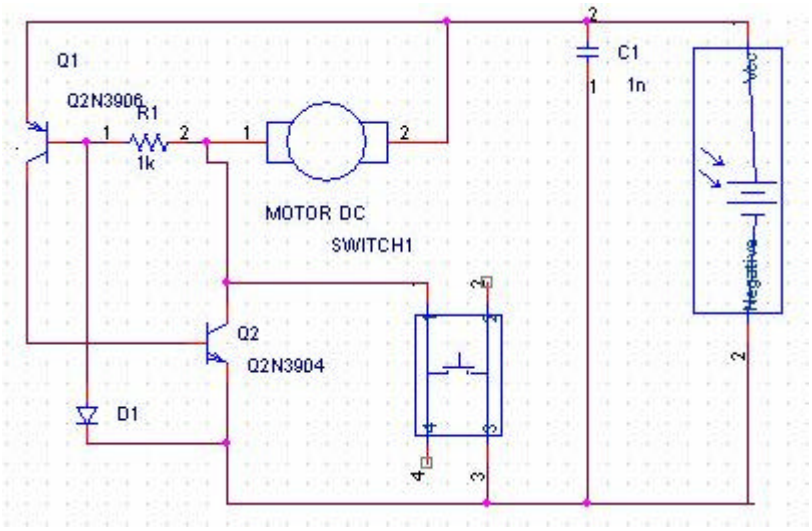


Figure 2-3. Diode-triggered drive circuit.

The basic operation of the circuit begins with the charging of the capacitor according to a time constant determined by the resistor and the capacitor. At the initial stages of charging, the current flowing into the capacitor is large relative to that of the current flowing through the motor, resistor, and diode. As the charging continues, the current flowing into the capacitor is reduced according to the time constant of the resistor and capacitor. Once a certain threshold voltage is reached at the base of the PNP transistor (Q1), the transistor turns on. At this point, current is allowed to flow from its emitter to collector. The collector current of Q1 flows to the base of the NPN transistor (Q2). The resultant voltage at the base of Q2 moves Q2 to the on state. As Q2 is turned on, the current through the motor is offered

less impedance to the negative rail. The current path changes to travel through Q2 rather than through the resistor and diode. This current flow drives the motor for a short time until the capacitor discharges to the point where Q1 and Q2 turn off. When both transistors are off, the charge cycle begins again.

2.3.2 Voltage detector-triggered circuit

The voltage detector-triggered circuit uses an MN1381L voltage detector to trigger the energy transfer from the capacitor to the motor as shown in Figure 2-4.

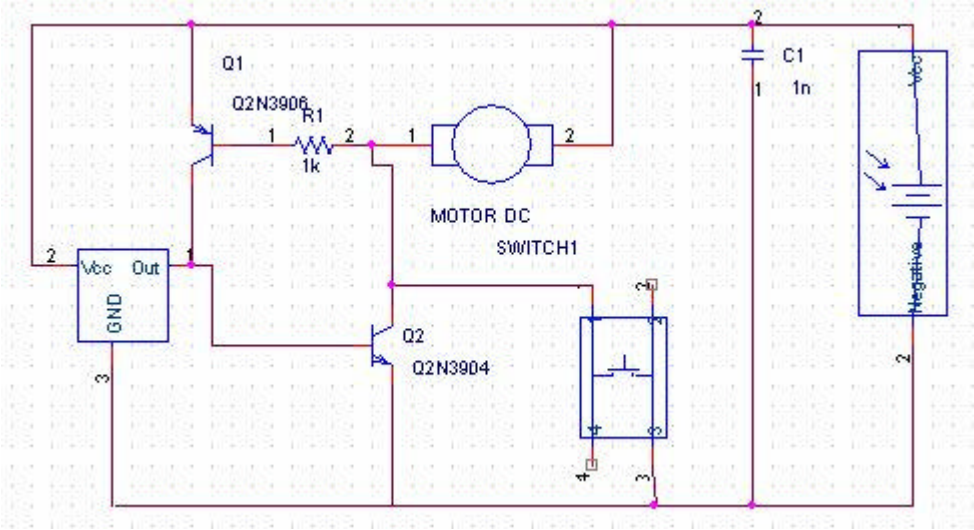


Figure 2-4. Voltage detector-triggered drive circuit.

When the solar cell charges the capacitor to a voltage in the range of 3.0 to 3.3 V, the base of the NPN transistor, Q2, is pulled high and current flows from the collector to the emitter of the transistor. The current drawn into Q2's collector causes the PNP transistor, Q1, to be in cut-off mode. The PNP transistor then allows current to flow from its emitter terminal to its collector and into the base of the NPN transistor. The PNP transistor thereby maintains the NPN transistor in the on state and allows the charge stored on the capacitor to flow through the motor on its way to the negative rail of the circuit. When the capacitor reaches the voltage level where the transistors cannot be maintained in the on state, approximately 0.7 V, the transistors turn off and current stops flowing through the motor. The solar cell voltage is then applied to the capacitor once again and the charging cycle begins anew.

The 3.0 V, or type L, version of the MN1381 voltage detector is used so that a relatively large voltage is available to drive the motor while keeping the demands on the solar cell reasonable. If a larger voltage is used, the charging time of the capacitor increases with little increase in motor torque. Conversely, a smaller voltage charges more quickly but yields no usable motor torque to drive the vehicle.

2.3.3 Voltage detector triggering vs. diode triggering

During qualitative testing in both bright sunlight and under fluorescent lamps it is observed that the charge cycle times and output torque of the motors are greater with the voltage detector-triggered drive circuit than with the transistor-triggered circuit.

Chapter 3

The Electronic Eye

Remote sensing of an unknown environment is an important role an autonomous vehicle can fulfill. An electronic eye is the chosen sensor for this project. The output that the eye produces is an array of analog voltage signals in the range of 0.0 to 5.0 V, corresponding to inputs of low to high light intensity, respectively. A phototransistor array is a suitable surveying mechanism to illustrate this concept and provide data to test the use of a Bluetooth connection.

3.1 Phototransistor Operation

Phototransistors are the same as a photodiode with the addition of a built in amplifier. Phototransistors have the property of larger current signals than photodiodes with relatively simple circuits. Figure 3-2 shows an enlarged view of a phototransistor chip. Photodiodes have unity gain with respect to input light intensity. As shown in Figure 3-1, the photons are absorbed the same way as a photodiode. The photocurrent is then fed into the base of a transistor. The transistor then amplifies the photocurrent proportional to β (usually 100-1000). The amplification of the photocurrent with the transistor is the same operation as a BJT transistor. The trade off for amplification is the slow response to light intensity. Typical transition for a phototransistor is 10μS, whereas in a photodiode the response is typically 0.5 μs. The slow response time is due to large parasitic capacitance in the transistor. The operating region in this design project, the average rises and fall time is 50μs. This does not affect our results as, we only want to sample every 20 ms.

With the base lead exposed, light sensitivity of the phototransistor can be adjusted. Phototransistors are linear over a 3.0 or 4.0 dB/decade range. As such, this project implemented this for a range that is in the normal light intensity of a room.

In phototransistors the signal and noise are amplified in the same proportion. Therefore the signal to noise ratio is the same as that found in a photodiode. In photodiode circuits, the major source of noise is thermal noise in the resistor due to the small signal created in the photodiode.

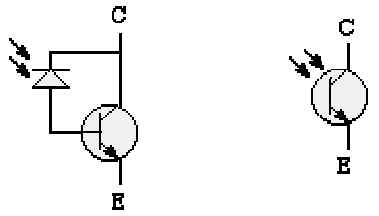


Figure 3-1. Circuit depiction of a phototransistor.

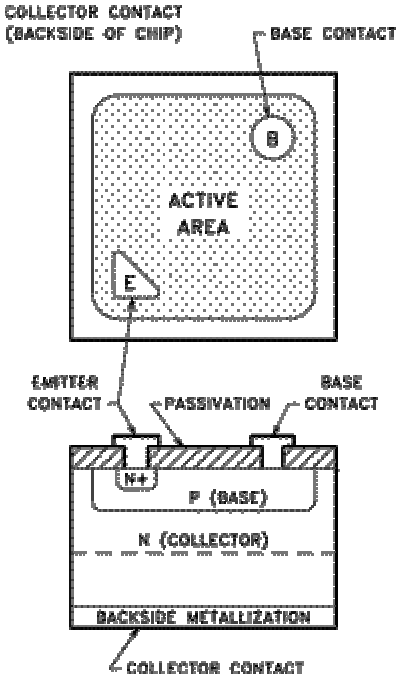


Figure 3-2. Detailed view of a phototransistor. [6]

3.2 Array Design

A phototransistor array is used to keep the size of the circuit small. Phototransistors require less supporting circuitry than photodiodes. In addition, phototransistors have high amplification of light intensity to voltage. Photoresistors are not used, as their response is too slow for this application. The physical layout of the phototransistors, shown in Figure 3-4, is optimized to get the biggest field of view. A limiting factor for how the array is arranged is the ability to identify erroneous information.

In this system, an array of sixteen phototransistors is used. In the design of the phototransistor array, demultiplexing is required. An analog sampler has a limited amount of analog inputs. Therefore, a multiplexing scheme is used. In order to simplify the demultiplexing circuit, an array of phototransistors with a size that is a power of two is used.

The circuit shown in Figure 3-4 is used to sample the light intensity. In order to have the phototransistor biased properly, an appropriate resistance value is chosen. Manufacturing techniques of phototransistors yield a variance of electrical characteristics at a given input light intensity. The output voltage changes due to efficiency of the detector, temperature, and wavelength of light. In order to achieve consistency throughout the array, adjustable resistors are used to adjust the bias of each phototransistor. A uniform input is used to standardize the output of the array.

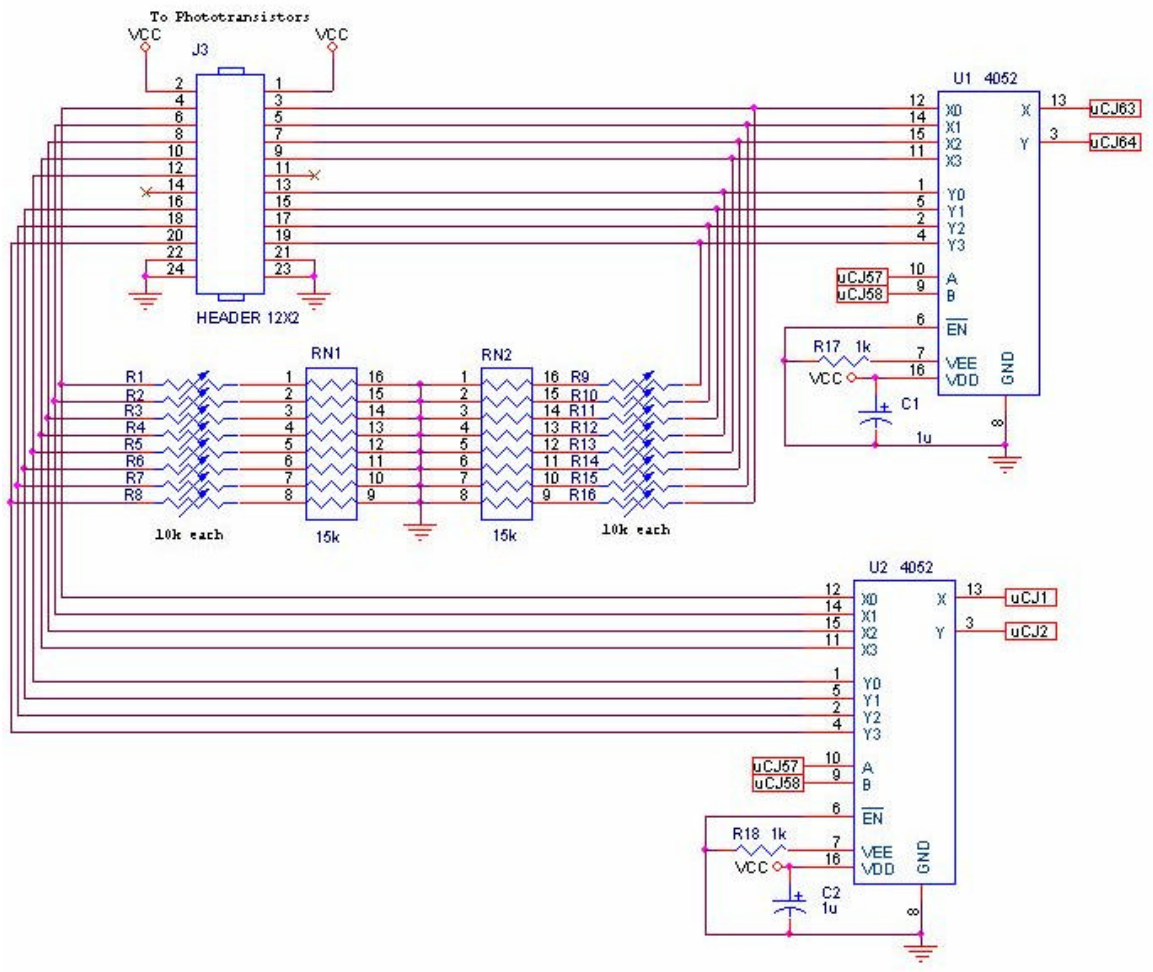


Figure 3-3. Phototransistor circuit on the microcontroller board.

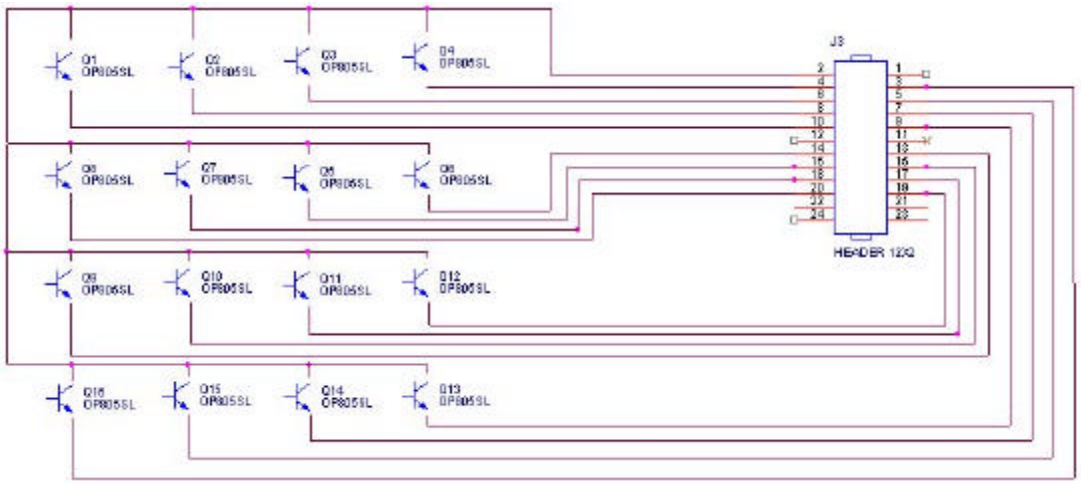


Figure 3-4. Phototransistor array circuit.

Maximum input impedance of the analog to digital converter (ADC) places another restraint on the sensor circuit. The data sheet for the analog to digital converter specifies its maximum input impedance to be 10 kΩ for proper operation. [10] The impedance of 10 kΩ chosen for operation is within the linear region of the phototransistor's operational curve.

3.3 Visualization

In order to view the information gathered from the phototransistors, image processing is recommended. Most of the image processing algorithms considered for this project use a mask size of three pixels by three pixels. [1] Therefore, only four of the sixteen pixels can be used for image processing. This results in large relative error per pixel of the image being displayed as only a quarter of the pixels are being processed.

To enhance contrast of the image, histogram equalization should be used. With such a small array of phototransistors, there are typically only small changes in the output voltage between each pixel. Therefore, not much information would be seen when displayed on a computer screen.

3.4 Board Layout

The most important consideration of the phototransistor array board layout is the physical size. The original proposal specifies a small autonomous vehicle that sends information to a base computer. The specification defines "small" to be a maximum of 20 cm vehicle in each spatial dimension. The phototransistors are mounted on a single board facing forwards on the vehicle and the required supporting hardware is carried on the main microcontroller board. Furthermore, having most of the components on the main board enhances the mechanical stability of the system.

Chapter 4

Circuit Designs

The schematic design was done in Orcad Capture and reviewed by all group members for availability parts at proper ratings. The schematics for the microcontroller board and Bluetooth board are contained in Appendix A and Appendix B respectively.

To provide modularity and circuit reusability, the hardware systems on the vehicle are divided between two printed circuit boards (PCBs). The first board holds the Ericsson ROK101008 Bluetooth module and its supporting circuitry. The second board houses the microcontroller and its supporting circuitry. AP Circuits manufactured these circuit boards.

4.1 Microcontroller Board

The supporting circuitry for the microcontroller board includes an RS-232 to 5.0 V serial communication interface, an interface for sampling data from the electronic eye and vehicle motors, a 5.0 V power supply circuit, and the majority of the electronic eye hardware. Figure 4-1 shows a block diagram of the microcontroller board.

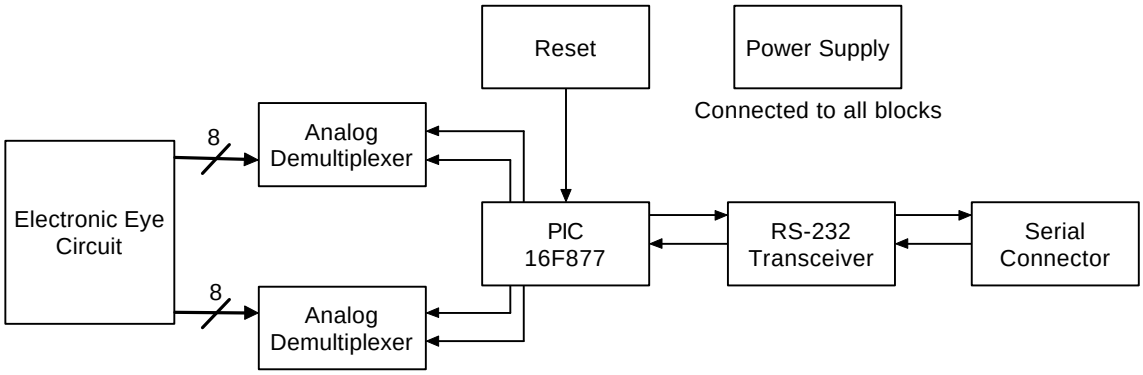


Figure 4-1. Microcontroller PCB block diagram.

The values and purposes of the components determine the working voltage ratings for capacitors and wattage ratings for resistors. Where required, power supply capacitors are rated to 25 V and 0.25 W resistors are used. These values allow for a safety margin for over voltages.

4.2 Bluetooth Board

A schematic of the Bluetooth board was created using the reference design from the manufacturer data sheet for the ROK101008. Once suitable part packages and values were determined, the schematic was updated with these part values.

Data sheets for the ROK101008 indicate that the Bluetooth module is designed for 3.3 V power supply. The supporting circuitry for the Bluetooth board includes a 3.3 V to RS-232 serial communication interface, a 3.3 V power supply circuit with reset switch for the Bluetooth module, and an 50 ohm antenna consisting of a copper trace on the printed circuit board. Figure 4-2 shows a block diagram of the microcontroller board.

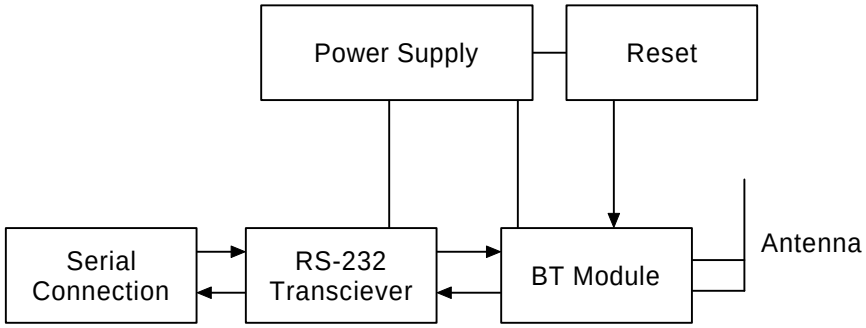


Figure 4-2. Bluetooth board block diagram.

A netlist for each schematic was created in Orcad Capture to allow the designs to be brought into the Layout software. The netlist is a tool used to determine component connections and is required by Layout to create a printed circuit board from the design schematic.

4.3 Component Creation

In Orcad Capture, each component requires a functional pin definition, which is a representation of the component's connections. Several Orcad component footprints were created for the design project including the Bluetooth module itself, capacitors, connectors, and voltage regulators. Each component was modeled to a limited extent in Orcad Capture. A separate footprint was then created in Layout to associate the padstacks with the pinout of the created component. The association of schematic components with layout footprints had to be done manually for each component made.

Packages were chosen according to ease of soldering, power dissipation requirements, and availability. Packaging options were noted during component creation for the layout work that would follow.

Each component's physical pins were created as padstacks in Orcad Layout using the built-in library manager and saved to several project-specific library files. The component data sheets were used for the physical dimensions of the component and padstack placement. For components for which data sheets could not be found, the components were measured. The part body and non-electrical details were placed on silk-screen layers for easier identification during part placement. The majority of the padstacks in each created component were designed as through-hole instead of surface mount. Through-hole components are attached to the board more easily than surface mount components with the technology available at the University of Manitoba.

An antenna is implemented as a trace on the board. This was done to eliminate the design work of implementing an off-board antenna.

4.4 Part Placement

The project proposal specifies an initial board size limit to work with. The physical width and length of the board are adjusted to contain all the parts of the board in their final positions while minimizing board size. Board size is affected by the limitation of the PCB manufacturer to use only two routing layers, namely top and bottom. Thus, a ground and power layer could not be used to simplify the PCB design.

Parts are placed in groups according to their connections to each other and these functional groups are placed according to their interconnectedness. These sections were identified in the schematic at the outset of part placement in Layout. In effect, the board consists of an assembly of interacting functional groups placed in their own sections on the board. For example, all components of the reset switch circuit are placed together including a transistor and capacitor to debounce the transient effects of the switch itself. The reset switch and related components were then placed together in one section of the board while the power supply components were placed in another section of the board.

The number of long connections between functional groups is minimized. Functional groups that have many connections to another functional group are logically placed next to each other. Placing related groups together leads to simplified routing and relatively fewer vias on the board. The time for the autoroute command to complete is significantly less with the functionally grouped board than without placing the components in functionally related sections. Reducing trace lengths between groups reduces the probability of noise and wave reflections on the data lines.

4.5 Trace Layout

The Orcad Layout autotrace feature was used for the initial layout after the part placement was finalized. Modifications and rerouting were done manually for each board where needed.

4.5.1 Trace width

A trace width of 10 mil was used throughout the design with deviation for the leads to the microstrip antenna that had trace widths of 39 mil to match the width of the antenna itself. The Bluetooth module's ball grid array posed several problems with routing and required extensive workarounds in the form of free vias that were placed under the pads of the device. The antenna transmission line trace width of 39 mil also offered a routing challenge. The pins of the Bluetooth module that drive the antenna are only slightly larger than the trace width required. The similar width of the ball grid array pads offered no way to make the transmission line to the antenna thicker than the antenna traces. In the final Bluetooth board design, the transmission lines were kept the same width as the antenna.

4.5.2 Trace spacing

A minimum spacing of 8.0 mil between traces is required by the manufacturer due to limitations of their board fabrication process. In the initial design sent to the manufacturer, a trace width of 6.0 mil was used for some traces. In some places, traces passed within 4.0 mil of the pads of the ball grid array of the Bluetooth module. The manufacturer rejected the initial design and a design utilizing trace widths of 8.0 mil with minimum 8.0 mil spacing were implemented shortly thereafter.

To avoid the spacing violations caused by running certain traces on the same layer as the ball grid array of the Bluetooth module, free vias are used under two pads of the Bluetooth module. This allows a 10 mil trace to be routed from each via on the bottom layer of the PCB instead of between pads of the ball grid array on the top layer of the PCB. These vias' drill holes are matched to the size of the Bluetooth module pad diameter. The free via copper dimensions on the top layer are also reduced to eliminate spacing violations.

4.6 Gerber File Creation

The Run Post Processor command in Orcad Layout creates several files, known as Gerber files, containing listings of information about the different layers of the board, physical dimensions, and drill hole locations. A file of apertures is created separately from the Orcad aperture spreadsheet. Once completed, the Gerber files were delivered to Ken Biegun, NT Lab Network Manager, who forwarded them to AP Circuits, the board manufacturer appointed to the project.

4.7 Board Assembly

The completed boards were received from AP Circuits four working days later. Parts were placed in their locations on the boards to check for physical errors. Several drill holes were found to be incorrect and would not properly fit the leads of some through-hole components. Those components that could not be easily inserted in the board were replaced or modified. Al McKay, of the tech shop, soldered the microcontroller printed circuit board.

It was theorized, at the time of the implementation of the under pad free via solution, that with a plated through-hole via of small enough drill hole diameter the via would fill with solder and offer a physically strong connection as well as an electrically stable one. Figure 4-3 shows the rationalization of the free via under pad solution.

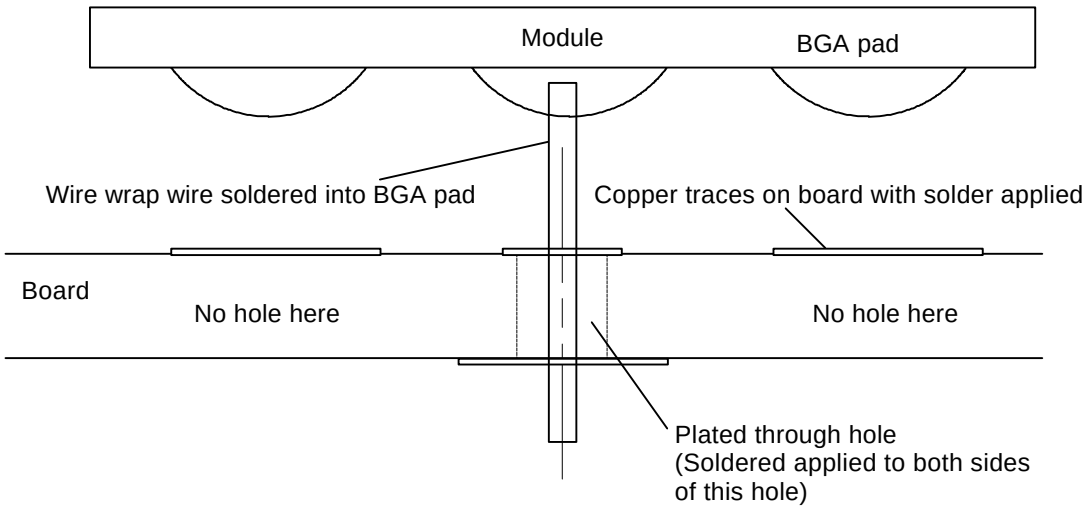


Figure 4-3. The free via under pad solution.

Chapter 5

The Bluetooth Module

The technology and protocol of Bluetooth is designed to be an inexpensive cable replacement that allows impromptu networking. Bluetooth enabled devices can seek connections with other Bluetooth enabled devices and automatically establish connections with them.

5.1 Ericsson Development Kit

5.1.1 General description and contents

The Ericsson Bluetooth Development Kit includes several tools useful in developing products that incorporate Bluetooth technology. The kit consists of an Ericsson ROK101008 Bluetooth module mounted on a printed circuit board with both RS-232 and USB ports. The sample software provided was written using Microsoft Visual C++ and demonstrates the use of the Bluetooth stack in a real world application using Microsoft's Common Object Modeling (COM) methodology.

5.1.2 Limitations on the project imposed by the kit

Several differences exist between the ROK101008 modules used in the development kits and commercial Bluetooth modules sold separately by Ericsson. The most significant difference is that the modules used in the development kits only support point-to-point piconets and cannot operate point-to-multipoint piconets.

As a possible expansion of the project is to develop multiple mobile surveyors, the capability for a multipoint piconet is desirable. Software design considerations would change for supporting a multipoint piconet.

Another limitation of the ROK101008 module is its inability to hold more than one command in its internal 'queue' at a time. The Bluetooth specification allows for a command stack on the module so that a command acknowledgment or completion is not required before issuing another command. The ROK101008 accepts only a single command and must issue a completion event before it can accept any other commands. The inability to buffer more than one command also placed a constraint on the design of the software.

The last limitation of the ROK101008 module is that it is not a production module. The ROK101008 is a module specifically designed for testing and development. As such, it is produced in limited quantities, it is relatively expensive, and it is difficult to find specifications for the module on the Ericsson web site. The documentation is distributed on the compact disc included in the development kit.

5.1.3 Development kit operation

The Bluetooth modules are capable of Universal Serial Bus and RS-232 serial communications. If USB is used, the USB driver for the Bluetooth module must be installed on the computer that communicates with the development kit. There are several problems with using a USB connection with the modules. The driver has compatibility problems with many USB hardware chip sets and does not install properly on several of the computers used. Teleca Comtec was contacted and their solution to the USB connection problem is to use the serial port to communicate to the Bluetooth module and use the USB for powering the development kit.

Using the application provided with the kit, there are several specific steps that must be executed in sequence in order to obtain a Bluetooth connection between two computers. The sequence that the software follows is outlined in the Bluetooth specification and consists of device discovery, service discovery, and service connection. The program must be run in the order indicated by the instructions found on the development kit's compact disc to correctly form a link. Once operational, the link performs well.

5.2 Bluetooth Technology Summary

The Bluetooth specification encompasses all aspects of networking. The protocol stack can be broken into a lower and an upper section as shown in Figure 5-1. The lower layers are responsible for the physical link and link control while the upper layers define logical links and service management. These upper layers are accessible by applications.

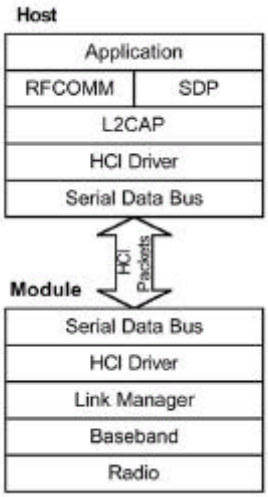


Figure 5-1. Bluetooth stack. [4]

The three lowest levels of the protocol stack primarily deal with the physical link. The Radio layer is the lowest and is responsible for the transmission and reception of data. The Baseband layer is responsible for all of the low level controlling functions. Physical channel definition, hop selection, channel control, and timing are the responsibilities of this layer. The Baseband layer is also responsible for Bluetooth addressing, error correction, and packet formation. The third lowest layer is the Link Manager. The Link Manager is responsible for setup and control of logical links. This layer is also the gateway to layers above it and filters out signals on the receiving side.

The next layer up the hierarchy, the Host Controller Interface (HCI) layer, is responsible for host and Bluetooth module communication as well as communication between the upper and lower sections. The HCI layer provides access to the hardware status registers and allows the host to control the behavior of the Baseband controller.

The next layer, the Logical Link Control and Adaptation Protocol (L2CAP), defines the connection state machine and performs the multiplexing of higher order links that allow more than one application or service to use the device.

The highest layer of the Bluetooth stack performs the presentation of the stack to the application. It includes the Service Discovery Protocol (SDP), a standardized method for listing services offered by the local device as well as listing services offered by Bluetooth devices in the local piconet. An example of one of the protocols used by applications is RFCOMM. RFCOMM presents a link that functions as a simple RS-232 serial link.

The protocol stack is typically implemented in one of two ways. The lower three layers, Radio, Baseband, and Link Manager, are usually embedded in the Bluetooth module along with an HCI interface layer. The host can implement the upper layers or they can be embedded in the module as well.

5.3 Specifications Implemented

Researching the Bluetooth specification led to the conclusion that the RFCOMM interface matched very well with the project. The ROK101008 module has only the lower layers embedded in the module and so requires the upper layers to be written in software. A suitable Bluetooth stack for the microcontrollers considered for the project is not presently obtainable. The criteria used for searching for a suitable stack are: i) the stack must have a small footprint in memory and ii) the stack must be free of cost and in the public domain. As a suitable stack could not be obtained, only the HCI commands necessary to allow for basic data communication are implemented in both the microcontroller and workstation applications.

5.3.1 HCI overview

A software developer writing a version of the HCI layer for an application needs to be aware of at least three types of packets that the host controller interface uses. The first type encountered is a command packet. The format of this packet type is shown in Figure 5-2. Command packets are used by the Bluetooth host to control the operation of the lower section of the stack embedded in a Bluetooth module (see Figure 5-1). The command set is mapped to two bytes of values. These values are known as the command opcodes, or operational codes. The minimum number of bytes following an opcode is one. This byte contains the number of bytes that follow it to convey any additional information required for the command.

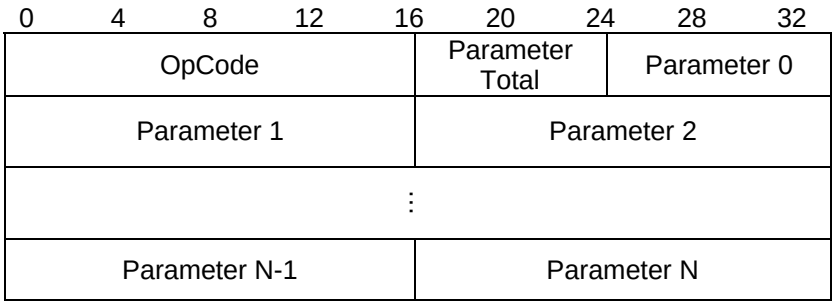


Figure 5-2. HCI command packet. [4]

The Bluetooth module responds to HCI commands with packets known as HCI events. It is also possible for the module to issue a HCI event to the application's HCI driver that isn't in response to a command. Figure 5-3 illustrates the general structure of a HCI event packet. As with command packets, each event is distinguished by the value of the first field, the event code. The total length field contains the number of bytes that follow it.

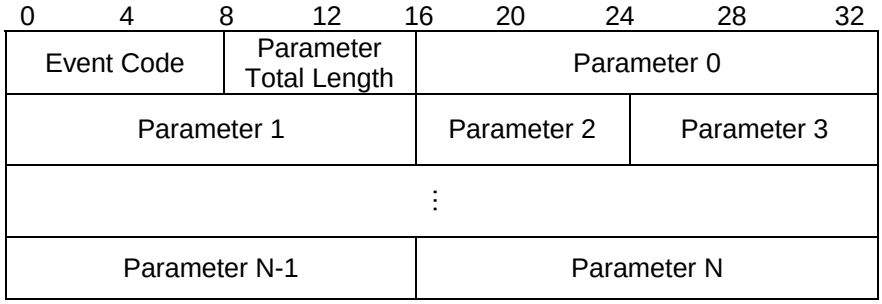


Figure 5-3. HCI event packet. [4]

Once a connection is established between two or more Bluetooth devices using the commands and events, the applications can transmit and receive data between each other. The transmission of data in this project is done using asynchronous, connectionless (ACL) data packets. A basic ACL data packet is structured as shown in Figure 5-4.

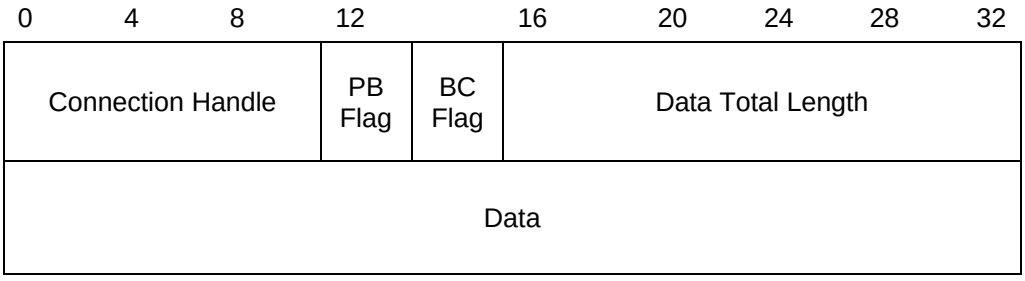


Figure 5-4. HCI ACL data packet. [4]

5.4 Antenna

5.4.1 Operational frequency

The Bluetooth module operates over 79 channels from 2.4 to 2.4835 GHz to allow for a secure and reliable connection in the Industrial, Scientific, and Medical (ISM) band. Bluetooth uses frequency hopping over the 79 channels to share the band, decrease the effect of noise, and increase the security of each link. A listener on the channel effectively hears random noise. The bits are mapped to a Gaussian Frequency Shift Keying (GFSK) encoding scheme allowing for greater noise tolerance.

5.4.2 Antenna implementation

The antenna is implemented as a trace on the Bluetooth module PCB. A width of 39 mil was used for both the antenna and the transmission lines to the antenna. This corresponds to a trace on the printed circuit board. The antenna length was calculated to be 3.125 cm for a quarter wavelength antenna.

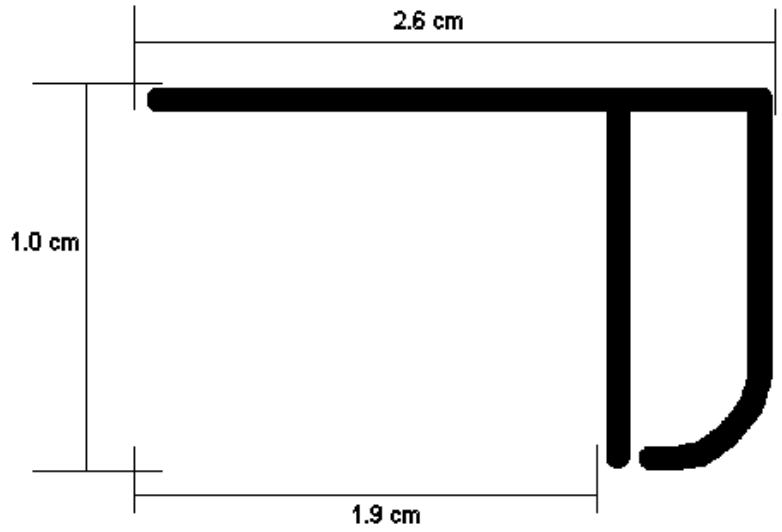


Figure 5-5. The microstrip antenna.

Chapter 6

Microcontroller Firmware

A microcontroller is used in this project to control a Bluetooth module residing on the surveyor as well a to perform analog to digital conversion of the signals from the electronic eye and motors. Several different microcontrollers were considered throughout the course of the project. At first, the Microchip PIC 16F877 was researched over the summer of 2001 but it was decided by the group to migrate to the Motorola MC68HC11 due to the unavailability of development tools for the PIC from within the department and the abundance of support tools for the 68HC11.

After finding that the 68HC11 did not have the capability to be electronically erased and reprogrammed in circuit, the search for an acceptable chip began again. In mid-January, it came to the group's attention that development tools for the PIC were available in the department at that time. At that point, the completed firmware for the MC68HC11, written in ANSI C, was ported to PIC 16877 assembly code. The actions of the firmware are shown below in Figure 6-1.

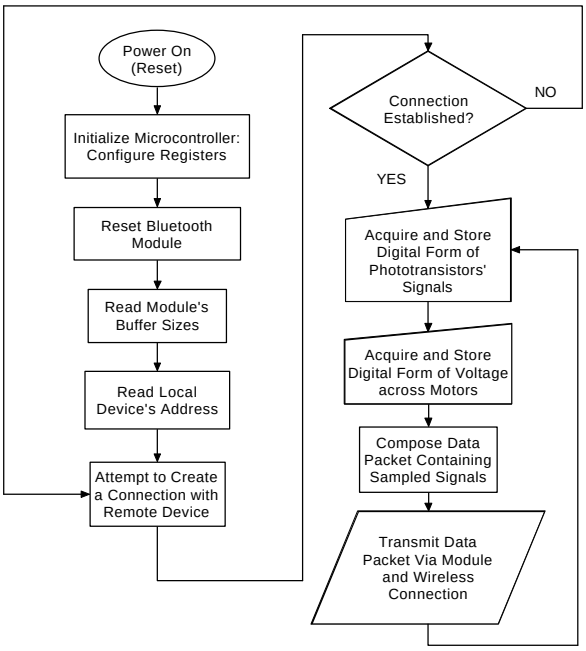


Figure 6-1. Micrcontroller firmware flowchart.

6.1 Microcontroller Initialization

The PIC 16F877 microcontroller has several registers that are configured at the beginning of the code written for this project to prepare the device for proper execution in the required states. The registers control the use of input and output pins, the analog digital converter module, the universal asynchronous receiver transmitter module, and interrupt handling. For more detail, please consult Appendix C and Reference 10.

6.2 Initialization of the Bluetooth Module

Though the full HCI layer is not implemented in the microcontroller software, certain commands are needed to prepare the connected ROK101008 module for use. As outlined in the Bluetooth module's data sheet released by Ericsson, HCI_Reset is the first command required. [12] An infinite loop is then entered to await the required response of the HCI_Command_Complete event. A status of zero is a success, while a non-zero value is a failure and is handled by re-issuing a HCI_Reset.

Once the Bluetooth module is reset, a HCI_Read_Buffer_Size command is expected. The response to this command, a generic HCI_Command_Complete event, contains the sizes of the buffers that the module has available for ACL and SCO data packets.

After the PIC program is aware of the buffer space that the module has available, a HCI_Read_BD_ADDR command is presented to the module The HCI_Command_Complete event used to respond to this command contains the unique six-byte Bluetooth device address of the attached module.

6.3 Establishing a Connection

Once the Bluetooth module is reset and its buffer size and address values are known, a connection to a remote Bluetooth device can be properly established. Under the Bluetooth specification, the normal procedure would be to send an inquiry message to get a list of other Bluetooth devices in range of the inquirer that are accepting connections. For the purpose of this project, however, the remote device's unique address is entered in the microcontroller program before compilation so that a request for connection can be sent directly to the specified remote device.

The command to initiate the procedure of negotiating a connection with a remote device is HCI_Create_Connection. This command's required parameters are: the address of the remote Bluetooth device, the packet type to use on the communication link, the page scan mode to use, the clock offset from one device to the other, a clock offset validity flag, and settings for whether a role switch is allowable.

The first response back to the microcontroller after this command is the HCI_Command_Status event. The local module uses this event to communicate to the microcontroller that the connection negotiation is, or is not, in progress. When the connection is established, the HCI_Connection_Complete event is transmitted to the microcontroller. As part of this event, the local module relays the connection handle, link type, and encryption mode associated with the connection.

With a valid connection handle in the microcontroller's memory, the software leaves the initialization stages and enters the perpetual loop of acquiring data, forming a data packet, and transmitting it over the connection.

6.4 Data Acquisition

The microcontroller addresses four of the sixteen phototransistors at a time by issuing the binary value 00, 01, 10, or 11 to the channel select pins of the two analog demultiplexors. After waiting 42 microseconds for the phototransistor output values to propagate through to the analog input pins of the microcontroller, the voltage level of each pin is successively sampled, converted to digital representation, and stored in memory. [13] After the first group of four phototransistor values is acquired, the addressing value is incremented to address the next four phototransistors. When the conversion of the fourth and final group's values is complete, the analog signals from the vehicle are similarly sampled, converted, and stored for transmission.

6.5 Data Packet Transmission

To send the acquired data to the remote Bluetooth device, an ACL data packet is formed. The packet is structured as shown in Figure 6-2 with the Connection Handle being obtained from the HCI_Connection_Complete event. Once the data packet has been transmitted through the microcontroller's serial module to the local Bluetooth module, the addressing of the phototransistor eye begins anew as described in Section 6.4.

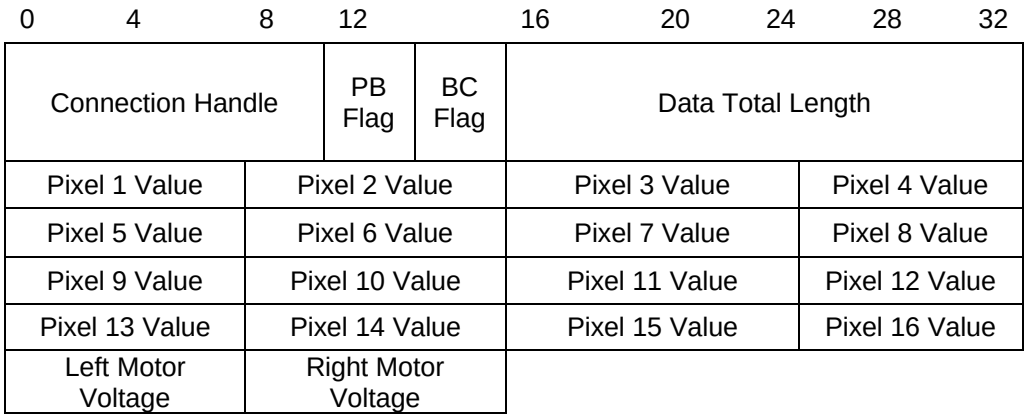


Figure 6-2. Surveyor data packet.

Chapter 7

Computer Interface and Bluetooth Stack

The design of the computer interface and implementation of portions of the Bluetooth stack required the design of several components in software at the graphical user interface and the individual bit levels. Several design choices needed to be made regarding what portions of the Bluetooth specification to implement and in the creation of objects that would eventually facilitate communication between the Bluetooth Development Kit and computer through a serial link. The computer interface aimed to be both versatile and resource light for fast data processing and real time image viewing.

7.1 Workstation Software Overview

7.1.1 Development environment

The environment used for software development was Microsoft Visual C++ 6.0. Visual Basic and LabView were also considered for a target environment. The sample Bluetooth code and reference stack was coded on the Microsoft Visual C++ compiler. The programmers were also familiar with the Visual C++ environment and had access to it.

7.1.2 Software development issues

The initial software for the workstation was to be developed using the reference stack provided in the Ericsson development kits. Modifications to the sample chat application were attempted to understand the functioning of the stack. The sample program and the stack are written to conform to the Microsoft Common Object Modeling (COM) specification. A version of the workstation software was written using the stack but debugging the code was difficult. The COM version was abandoned due to the length of time estimated to fix all the errors. The reference stack also is more complicated than the software written for the microcontroller. Due to the difficulties with the development using the reference stack and our desire to enable the microcontroller software to be simulated in the Visual C++ environment we decided to implement a version of the HCI layer as was done in the microcontroller code. As developing and testing code is easier in the Visual C++ environment, the microcontroller code was to be tested first on the workstation reducing the debug time.

There were several problems involved in the development of the simplified HCI driver code. Microsoft Foundation Class (MFC) was the basis for the driver developed. The largest problem encountered was the lack of proper support in the MFC for serial communications. A free wrapper class was obtained to try to work around the lack of support, but unfortunately the problems with serial communications persisted and stopgap measures had to be implemented.

In the code snippet below, the DataWaiting method is expected to return a value of true when data is present at the serial port however it doesn't function properly. The Pause function had to be inserted to allow time for the Bluetooth module to return an event after the application issues a HCI command.

```
TheModule.WriteToModule(HCI_READ_BD_ADDR, m_SerialPort);
Pause(20); //wait for 20 milliseconds for data to be ready.
BytesToRead = m_SerialPort.BytesWaiting();
if (BytesToRead == 0){
        WaitResult = m_SerialPort.DataWaiting(INFINITE);
        if(!WaitResult) {
                AfxMessageBox("Waiting timeout for Reading BD Address!");
                OnCancel();
        }
```

7.1.3 Software description

The software was functionally decomposed into two classes. The Bluetooth stack was implemented as one class. The Bluetooth class handles connection setup and data handling. The application was written to display the data in a graphical user interface as well as store the data received in a log to allow for asynchronous analysis of the collected data.

7.2 Initialization of the Bluetooth Module

The standard Bluetooth connection process to connect to a service is to search for Bluetooth devices (Inquire), connect to the discovered device (Page), discover what services the device supports (Service Discovery Protocol), decide which service to connect to (Application), learn how to connect to the chosen service (Service Discovery Protocol) and finally connect to the service (Application). Figure 7-1 illustrates the workstation software's flow of control from initialization to connection formation, data transmission and finally to a possible connection breakdown.

We decided to shorten the connection process for simplicity. Our streamlined connection process is to immediately connect to the device and the only service that will be offered on the computer is data handling from the eye. This type of connection is called a bonded pair and is used primarily to connect two devices, which form a unique two-way communication pair. [5]

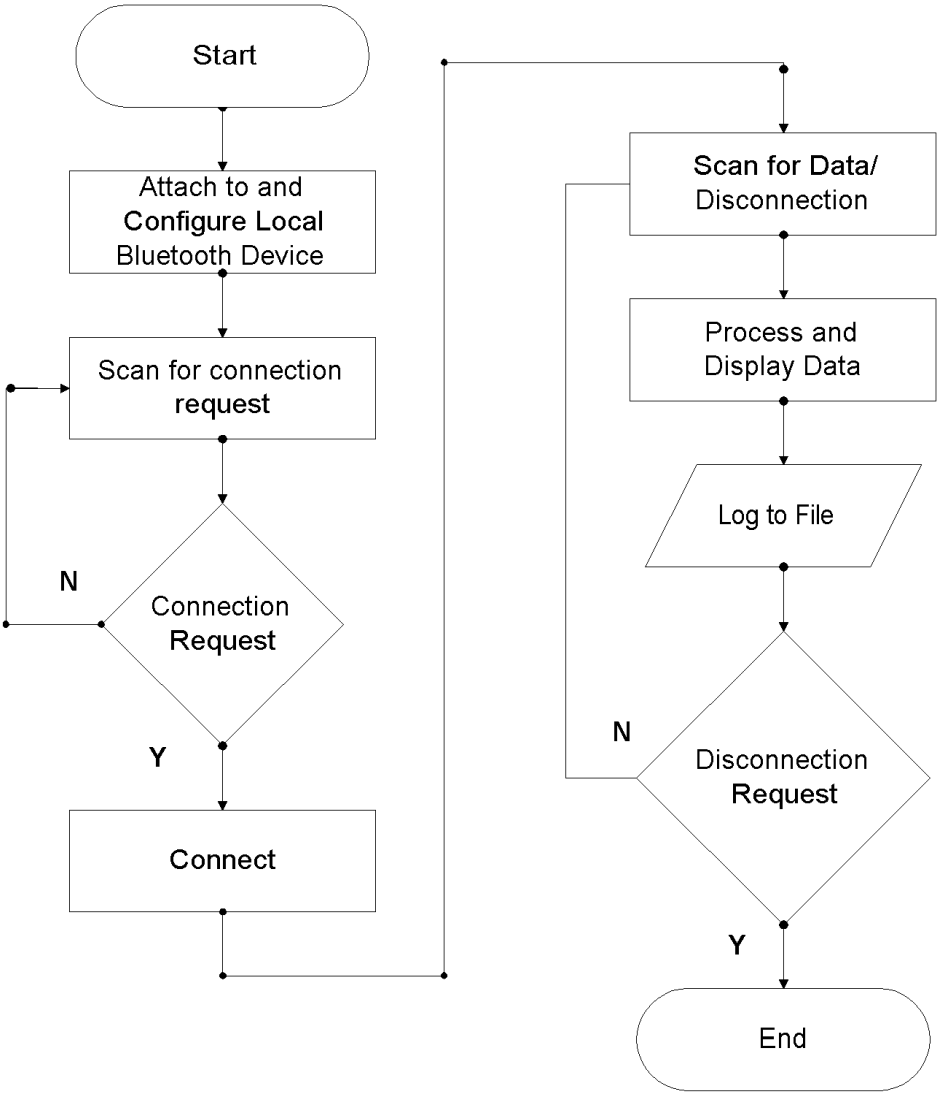


Figure 7-1. Computer software flowchart.

The software initialization begins by opening the log file and initializing the file by writing the current date and time to the file header. Next, the serial port is opened and initialized to the specified settings to communicate to the Bluetooth module. HCI_Reset is required to be the first command issued to the Bluetooth module and is the first command sent to the module. [12] The software will then wait for a HCI_Command_Complete event indicating a successful reset before continuing. The program will continue to retry to reset the module for a limited number of times. If all of the retries are exhausted the program reports an error and exits. Once successfully reset, the local Bluetooth address is read using HCI_Read_BD_Addr. The local address is then displayed on the graphical user interface (GUI) to indicate a successful connection with the local module. The Bluetooth module is then set up to automatically accept a connection by using the HCI_Set_Event_Filter command. After the successful return of a HCI_Command_Complete event, the module is then placed in Page Scan Mode using HCI_Write_Scan_Enable to listen for a connection request from the vehicle.

7.3 Connection and Data Reception

The software waits until the vehicle requests a connection. The module will automatically accept the connection and will issue a HCI_Connection_Complete event to indicate a successful connection. The HCI_Connection_Complete event passes the active connection handle to the application. For completion, the software then disables Page Scan Mode to save on power and because it cannot accept any further connections.

With the Bluetooth connection established, data is sent from the vehicle to the computer using ACL data packets. The software then waits for data packets or for the vehicle to close the connection. On reception of a data packet, the software stores the data to a log file and displays it on the screen. The software then continues to wait for data packets. When data transmission is complete the vehicle can issue a HCI_Disconnection command. The module returns a HCI_Disconnection_Complete event, which passes the reason for disconnection and the connection handle of the disconnected channel.

7.4 Graphical User Interface

The interface, seen in Figure 7-2, is split into eye, vehicle, and connection information sections. The visual data received from the vehicle eye is displayed in a four by four matrix of squares with each element of the matrix representing a phototransistor of the electronic eye. Each element displays the output of the phototransistor with a 256 level grayscale image. The interface also indicates the voltage across the motors. The direction of the motors is on the dialog box but has not been implemented. The Bluetooth address of the module attached to the workstation is displayed on the title bar and the address of the remote address is indicated on the Bluetooth Address output box.

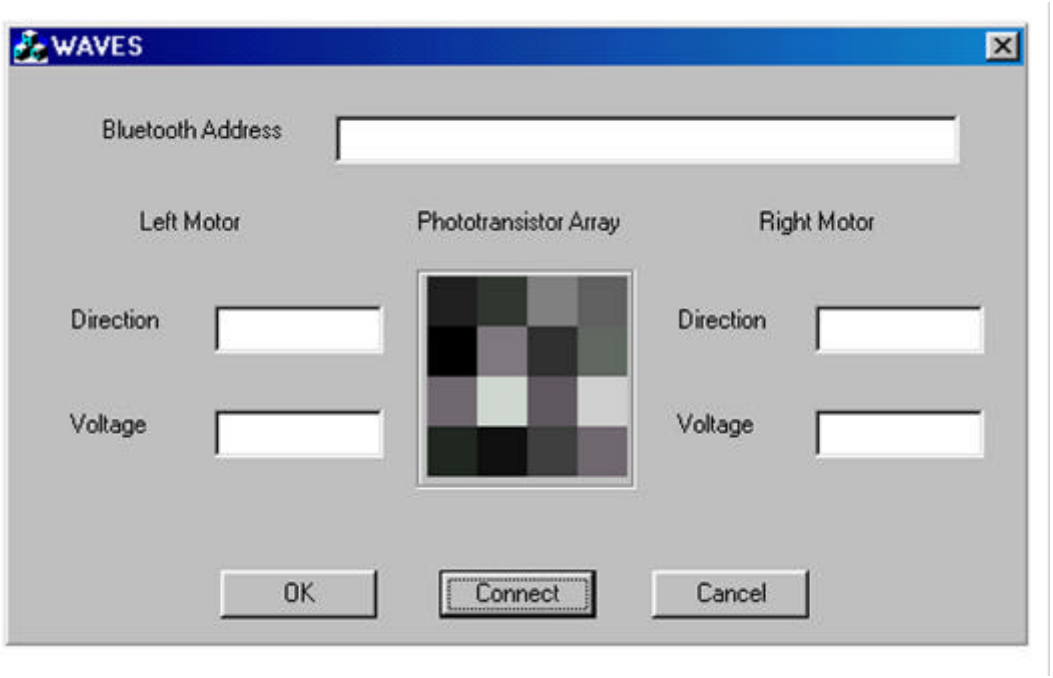


Figure 7-2. Computer application screenshot.

Chapter 8

Conclusions

8.1 The Vehicle

The drive circuits were successfully constructed and found to be efficient enough to be able to move the vehicle in bright sunlight, but not under indoor lighting. The chassis proved straightforward to design and implement and was not overly bulky. Research into a power supply that would allow the vehicle to operate in the dark by maintaining a battery could be done to further the effectiveness of the vehicle in an unknown environment. Also, conversion of the remaining circuits carried by the vehicle to solar power would greatly increase the usefulness of the system for operation in remote areas.

8.2 The Electronic Eye

The electronic eye was successfully constructed but not tested. The physical construction of the eye circuitry was completed but, due to problems in other circuitry, the phototransistor array could not be tested. The Bluetooth board was not completed due to the inability to properly solder the ball grid array of the Bluetooth module. Preliminary tests were performed by attaching an oscilloscope to the output of some phototransistors and the operation was successful. Therefore, if the circuitry was working properly, it is felt that the phototransistors would perform as expected.

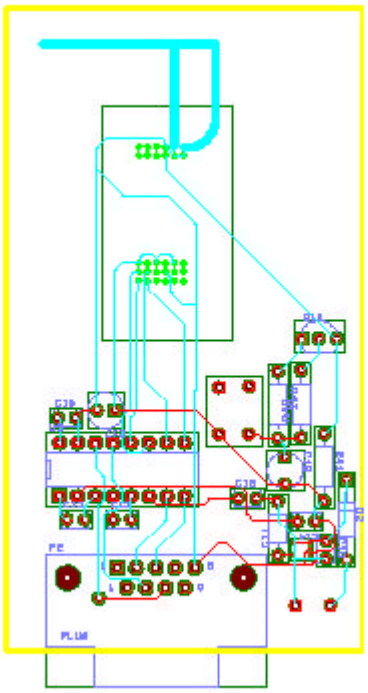
8.3 The Microcontroller Firmware

The microcontroller firmware worked successfully in simulation of the code and the algorithms used in the firmware worked when ported over to another Windows-based machine and coded in Visual C++. The firmware was not tested on the microcontroller itself as the microcontroller printed circuit board was not fully functional at the time of the project's completion. Had the circuit been breadboarded rather than designed in Orcad Layout, etched, and soldered, the time saved by not having to experience the learning curve of Orcad Layout would have been enough to fully complete the project. That said, the time spent learning Orcad gave the team valuable experience for future circuit board design projects.

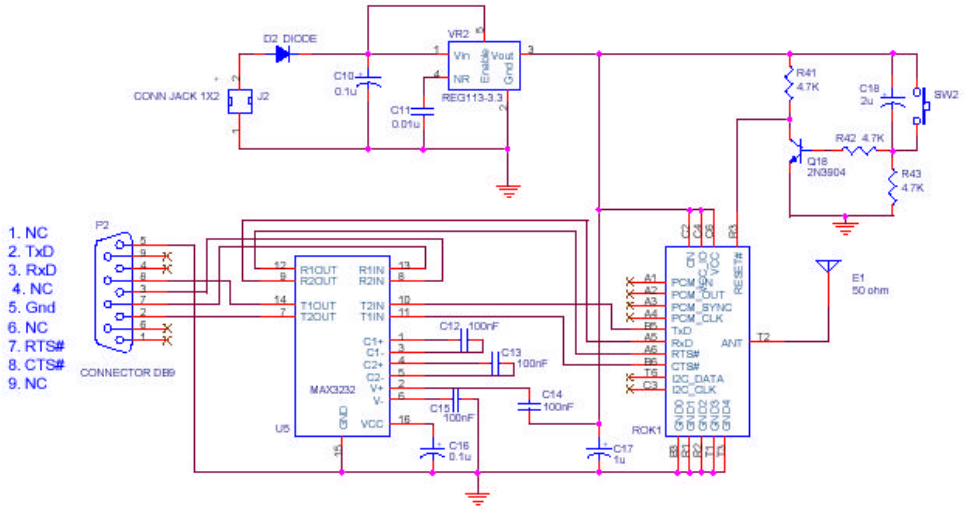
8.4 The Workstation Software

The HCI driver implemented worked as expected. The application also connects as expected during the preliminary test completed, however there was no test done to find the limit at which data can be received. The major recommendations are: Changing the development environment used to Visual Basic(VB). It was learned after consulting experienced VB programmers that VB handles serial communication properly and is a faster development environment. It is also suggested to implement enough of the Bluetooth stack to use RFCOMM to communicate to the robot and the vehicle. RFCOMM would present the application with a much simpler interface to the remote sensor making it easer to write, test and modify the code written.

Appendix A Bluetooth Board Schematic and Layout



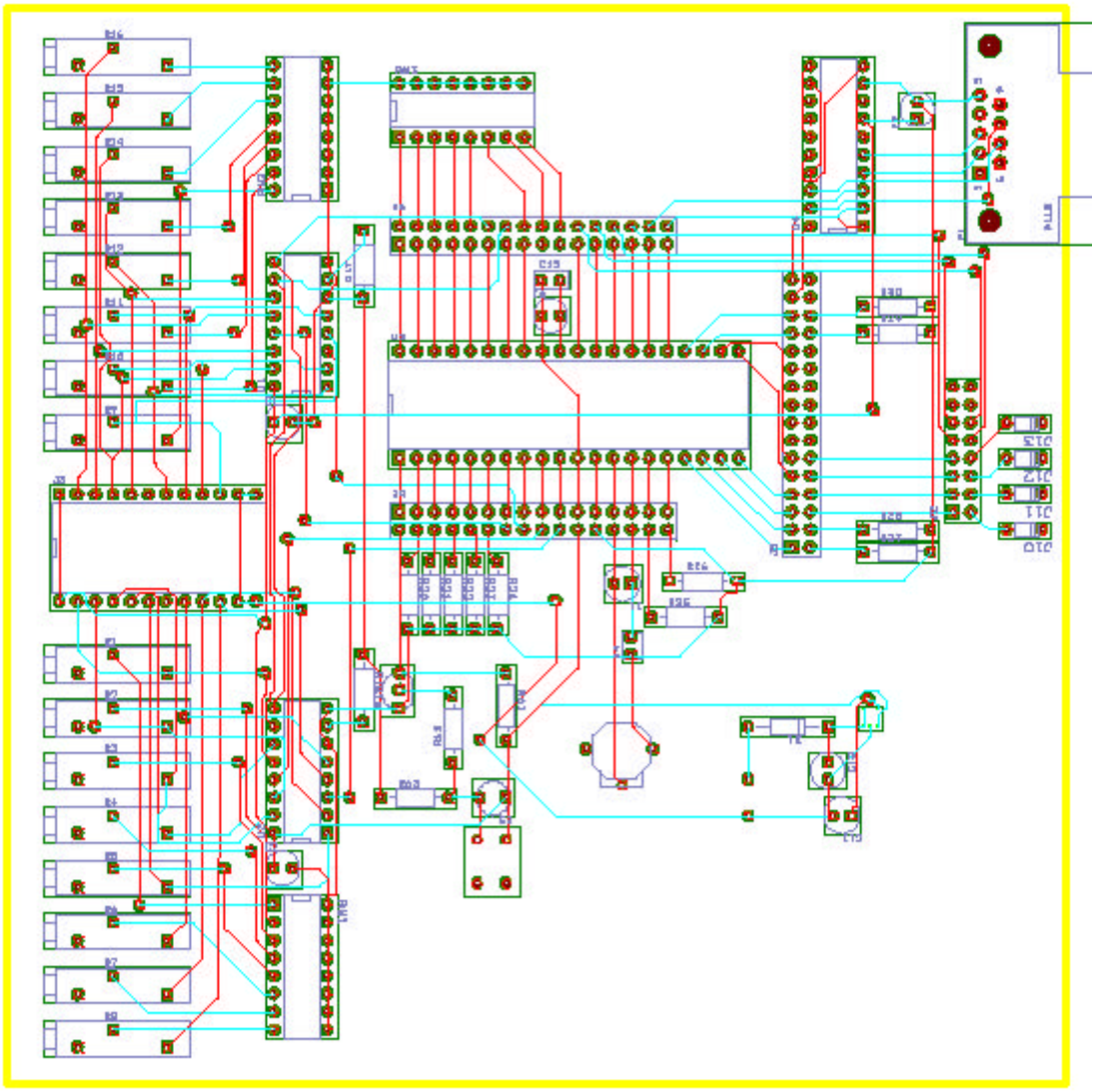
Bluetooth Layout



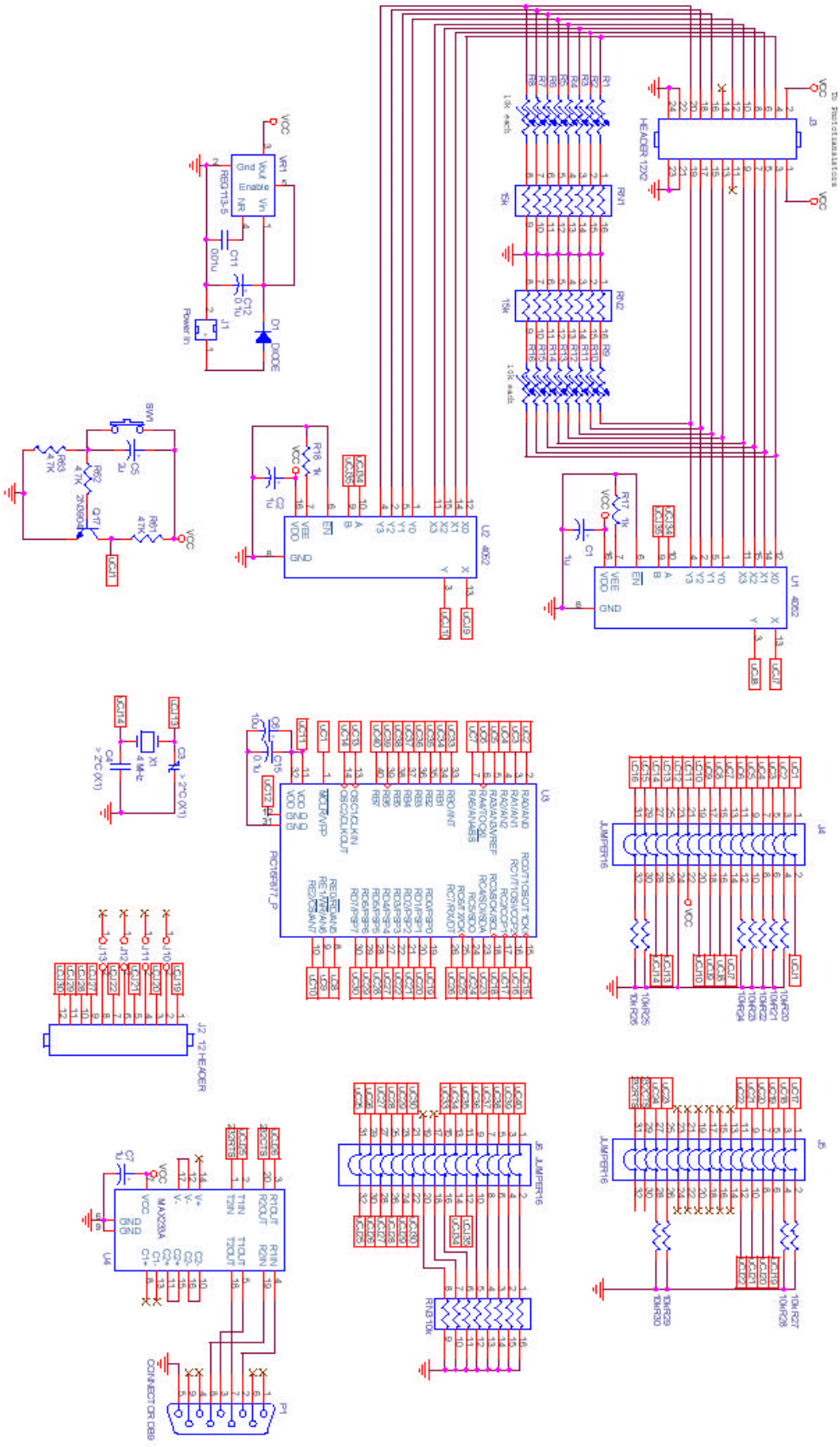
Bluetooth Layout Top View

Bluetooth Layout Bottom View

Appendix B Microcontroller Board Schematic and Layout



Microcontroller Board Layout



Microcontroller Board Schematic

Microcontroller Layout Top View

Microcontroller Layout Bottom View

Appendix C PIC 16F877 Microcontroller Firmware Listing

```
;*****************************************************************************
; BTBLITZ.ASM
;  Read in data from 16 phototransistors, chassis sensors,
;   motor control lines, and send them out the serial pins
;   to a Bluetooth (BT) module

    LIST P=16F877
    include "p16f877.inc"

;-----------------------------------------------------------------------------
; Constants
;-----------------------------------------------------------------------------

; base frequency
#define XTAL_FREQ           3686400       ;OSC freq in Hz
;Note: an instruction cycle = 4*(1/XTAL_FREQ), in our case, 1.0851 uS
;      Some instructions take 2 cycles
; buffers for serial communication (best when size = power of 2)
#define RX_BUFFER_SIZE      32            ;serial receive buffer allocation
#define TX_BUFFER_SIZE      32            ;serial transmit buffer allocation
#define DATA_BUFFER_SIZE    32            ;ACL data packet buffer allocation
#define MAXEVENTSIZE        16
#define ACLData             2
#define EVENT               4
; calculates baudrate when BRGH = 1, adjust for rounding errors
;#define CALC_HIGH_BAUD(BaudRate) (((10*XTAL_FREQ/(16*BaudRate))+5)/10)-1
#define CALC_HIGH_BAUD(BaudRate)          (XTAL_FREQ/(57600*16))-1
; calculates baudrate when BRGH = 0, adjust for rounding errors
;#define CALC_LOW_BAUD(BaudRate)    (((10*XTAL_FREQ/(64*BaudRate))+5)/10)-1
#define CALC_LOW_BAUD(BaudRate)           (XTAL_FREQ/(57600*64))-1
; ************* Bit variable definitions **************************
#define _BufferOverrun      COM_Flags,0
; ****************** * Predefined strings to send by uart *****
#define HCI_Reset             0x00        ; See STRING0 at end of listing
#define HCI_Read_Buffer_Size  0x01        ; See STRING1 at end of listing
#define HCI_Host_Buffer_Size  0x02        ; See STRING2 at end of listing
#define HCI_Read_BD_ADDR      0x03        ; See STRING3 at end of listing
#define HCI_Create_Connection 0x04        ; See STRING4 at end of listing

;-----------------------------------------------------------------------------
; Variables - RAM space set to 0x20 to  0x7F (Bank 0)
;                               0xA0 to  0xEF (Bank 1)
;                              0x110 to 0x16F (Bank 2)
;                              0x190 to 0x1EF (Bank 3)
;          Addresses 0x00 to 0x1F, 0x80 to 0x9F, 0x100 to 0x10F, 0x180 to 0x18F
;                 are for Special Function Registers
;-----------------------------------------------------------------------------
  ; *** Bank0 *** 80 bytes
  CBLOCK  0x020
;  Temp:1                ; temp byte ONLY to be used locally and no calls
  SendStrIndex:1        ; Used as an index into the string table (temp)
  SendStrTmp:1          ; Save a copy of which string we're to write (temp)
  ; serial buffer RX and TX  pointers, buffers located in bank1
  RX_Buffer_InPtr:1     ; where to put next incoming byte
  RX_Buffer_OutPtr:1    ; where to get next ( first ) byte
  RX_Buffer_Count:1     ; how many we have in buffer
  TX_Buffer_InPtr:1     ; where to put next outgoing byte
  TX_Buffer_OutPtr:1    ; where to get next byte
  TX_Buffer_Count:1     ; how many we have in buffer
  TX_Temp:1             ; temporary reg used while sending
  COM_Flags:1           ; flags for serial communication
  ENDC
  ; *** Bank0/1/2/3 mirrored in all banks 0x70, 0xF0, 0x170, 0x1F0, 16 bytes
  CBLOCK  0x070
  ICD_Reserved1:1       ; for icd (in circuit debugging)
  ; ram variables accessible from all banks mainly used for context saving
  Saved_W:1             ; variable used for context saving
  Saved_Status:1        ; variable used for context saving
  Saved_Pclath:1        ;
  Saved_Fsr:1           ;
  Table_Temp:1          ; table lookup temp variable
  NumHCICommandPackets:1;arbitrary starting value(10?), updated by Host Controller
  RequestStatus:1       ;variable to hold status of commands temporarily
  LoopI:1               ;temp variable for "For" loops
  DataLength:2          ;used for rx/tx of data packets
  btemp:1               ;
  ENDC
  ; *** Bank1 *** 80 bytes
  CBLOCK  0x0A0
  RX_Buffer:RX_BUFFER_SIZE      ; buffer for receiving
  TX_Buffer:TX_BUFFER_SIZE      ; buffer for sending
  ENDC
  ; *** Bank2 *** extra ram 16 bytes
  CBLOCK  0x110
  VisionArray:16                ;Reserving 16 bytes at this time, to 0x11F
  ENDC
  ; *** Bank2 *** 80 Bytes
  CBLOCK  0x120
  SensorStates:1                ;Holds data read from simple sensors
  MotorStateFrLf:1              ;Holds last read value of the FL motor
  MotorStateFrRi:1              ;Holds last read value of the FR motor
  MotorStateBkLf:1              ;Holds last read value of the BL motor
  MotorStateBkRi:1              ;Holds last read value of the BR motor
  HC_ACL_Data_Packet_LengthL:1  ;updated by HCI_Read_Buffer
  HC_ACL_Data_Packet_LengthH:1  ; and contain the host controller's
```

```
  HC_SCO_Data_Packet_Length:1   ; data buffer information
  HC_Total_Num_ACL_Data_PacketsL:1 ;
  HC_Total_Num_ACL_Data_PacketsH:1 ;
  HC_Total_Num_SCO_Data_PacketsL:1 ;
  HC_Total_Num_SCO_Data_PacketsH:1 ;
  BD_ADDR_local_1:1    ;Obtained by HCI_Read_BD_ADDR,
  BD_ADDR_local_2:1    ; this string is the address for
  BD_ADDR_local_3:1    ; this specific Bluetooth radio
  BD_ADDR_local_4:1    ; (manufacturer assigned)
  BD_ADDR_local_5:1    ;
  BD_ADDR_local_6:1    ;
  BD_ADDR_remote_1:1    ;Obtained by <edit>,
  BD_ADDR_remote_2:1    ; this string is the address for
  BD_ADDR_remote_3:1    ; the Bluetooth radio that we are connected to
  BD_ADDR_remote_4:1    ; or will connect with
  BD_ADDR_remote_5:1    ;
  BD_ADDR_remote_6:1    ;
  Conn_HandleL:1         ;ACL connection handle
  Conn_HandleH:1         ;  obtained by Create_Connection_Complete
  Link_Type:1            ;
  Encryption_Mode:1      ;
  Packet_Type:1               ;
  PacketBuff:16               ;
  ENDC
  ; *** Bank3 *** extra ram 16 bytes
  CBLOCK  0x190
  ENDC
  ; *** Bank3 *** 80 bytes
  CBLOCK  0x1A0
  ACLDataPacket:DATA_BUFFER_SIZE  ;Memory space to compose the data packet in
;  ACLEventPacket:DATA_BUFFER_SIZE ;Memory space to parse the event packet from
;  ACLDataPtr:1                ;where to put next byte of data
;  ACLEventPtr:1                ;where to put/read next byte of event to/from
  ENDC
  CBLOCK  0x1EB
  ICD_Reserved2:5              ; for icd
  ENDC

; ***************** Macro definitions ******************************
;+++++
; PAGE/BANK0/1/2/3 selects register bank 0/1/2/3.
; Leave set to BANK0 normally.
BANK0 MACRO
  BCF       STATUS,RP0      ; clear bank select bits
  BCF       STATUS,RP1
  BCF       STATUS,IRP      ; clear indirect addressing bit
  ENDM
BANK1 MACRO
  BSF       STATUS,RP0      ;
  BCF       STATUS,RP1      ;
  BCF       STATUS,IRP      ; clear indirect addressing bit
  ENDM
BANK2 MACRO
  BCF       STATUS,RP0      ;
  BSF       STATUS,RP1
  BSF       STATUS,IRP      ; set bit for indirect addressing
  ENDM
BANK3 MACRO
  BSF       STATUS,RP0      ;
  BSF       STATUS,RP1
  BSF       STATUS,IRP      ; set bit for indirect addressing
  ENDM
PAGE0 MACRO
  BCF       PCLATH,3
  BCF       PCLATH,4
  ENDM
PAGE1 MACRO
  BSF       PCLATH,3
  BCF       PCLATH,4
  ENDM
PAGE2 MACRO
  BCF       PCLATH,3
  BSF       PCLATH,4
  ENDM
PAGE3 MACRO
  BSF       PCLATH,3
  BSF       PCLATH,4
  ENDM

;+++++
; TABLE_JUMP Calculates an eventual page boundary crossing
; set's up the PCLATH register correctly
; Offset must be in w-reg, offset 0 jumps to the next instr.
; Uses one byte of dedicated ram
TABLE_JUMP  MACRO
  MOVWF     Table_Temp      ; save wanted offset
  MOVLW     LOW($+8)        ; get low address ( of first instr. after macro )
  ADDWF     Table_Temp,F    ; add offset
  MOVLW     HIGH($+6)       ; get highest 5 bits (of first instr. after macro)
  BTFSC     STATUS,C        ; page crossed ? ( 256 byte )
  ADDLW     0x01            ; Yes add one to high address
  MOVWF     PCLATH          ; load high address in latch
  MOVF      Table_Temp,W    ; get computed address
  MOVWF     PCL             ; And jump
  ENDM

;+++++
; PUSH/PULL save and restore W,PCLATH,STATUS and FSR registers -
; used on interrupt entry/exit
PUSH  MACRO
  MOVWF     Saved_W         ; save w reg
```

```
  SWAPF     STATUS,W        ;The swapf instruction, unlike the movf, affects NO status bits, which is why it is
used here.
  CLRF      STATUS          ; sets to BANK0
  MOVWF     Saved_Status    ; save status reg
  MOVF      PCLATH,W
  MOVWF     Saved_Pclath    ; save pclath
  CLRF      PCLATH
  MOVF      FSR,W
  MOVWF     Saved_Fsr       ; save fsr reg
  ENDM
PULL  MACRO
  MOVF      Saved_Fsr,W     ; get saved fsr reg
  MOVWF     FSR             ; restore
  MOVF      Saved_Pclath,W  ; get saved pclath
  MOVWF     PCLATH          ; restore
  SWAPF     Saved_Status,W  ; get saved status in w
  MOVWF     STATUS          ; restore status ( and bank )
  SWAPF     Saved_W,F       ; reload into self to set status bits
  SWAPF     Saved_W,W       ; and restore
  ENDM

;+++++
; INC_BUFFER advance buffer pointers wrap if necessary
;
; If buffer size is power of two, and buffer is aligned
; on an multiple of twice it size, this macro generates
; two instructions, Otherwise it generates six instructions.
; Originator:  Eric Smith, eric@brouhaha.com for non-commercial use.
INC_BUFFER MACRO  Pointer, Base, Size
  LOCAL POWER_OF2, ALIGNED,BIT,VALUE
POWER_OF2 SET !(Size&(Size-1))  ; calculate if power of 2
ALIGNED   SET POWER_OF2&&((Base&(Size-1))==0) ; calculate if aligned
  IF  ALIGNED
VALUE SET Size
BIT SET 0
  WHILE VALUE>1
BIT SET BIT+1
VALUE SET VALUE>>1
  ENDW
  ENDIF
  INCF      Pointer,F           ; increase pointer
  IF  ALIGNED&&!(Base&(1<<BIT)) ; aligned ?
   BCF Pointer,BIT              ; yes, clear bit
  ELSE
   MOVF     Pointer,W           ; no
  XORLW     Base+Size
  MOVLW     Base
  BTFSC     STATUS,Z
  MOVWF     Pointer
  ENDIF
  ENDM

;+++++
; DISABLE_IRQ disable global irq
DISABLE_IRQ MACRO
  LOCAL STOP_INT
STOP_INT
  BCF       INTCON,GIE          ; disable global interrupt
  BTFSC     INTCON,GIE          ; check if disabled
  GOTO      STOP_INT            ; nope, try again
  ENDM

;+++++
; ENABLE_IRQ enable global irq
ENABLE_IRQ MACRO
  BSF       INTCON,GIE          ; enable global interrupt
  ENDM
; ****************** END macro definitions **************************

;-------------------------------------------------------------------------
;   Set the reset vector here.
;-------------------------------------------------------------------------
  ORG       0x00
  NOP                           ; required for the ICD
  CLRF      STATUS              ; ensure we are at bank0
  CLRF      PCLATH              ; ensure page bits are cleared
  GOTO      INIT

;-------------------------------------------------------------------------
; Interrupt handler
;  also: Set the interrupt vector here
;-------------------------------------------------------------------------
  ORG       0x04
  PUSH                          ; save registers
INT
  ; Interrupt code
INT_TEST_RX_IRQ
  BTFSS     PIR1,RCIF           ; test if serial receive irq
   GOTO      INT_TEST_TX_IRQ     ; nope check next
  ; rx irq
  CALL      RX_INT_HANDLER      ; goto rx handler
  BCF       PIR1,RCIF           ; clear rx int flag
INT_TEST_TX_IRQ
  BTFSS     PIR1,TXIF           ; test if serial transmit irq
   GOTO      INT_TEST_NXT
  ; tx irq
  CALL      TX_INT_HANDLER      ; goto tx handler
  BCF       PIR1,TXIF           ; clear tx int flag
INT_TEST_NXT
  ; test whatever left
INT_EXIT
  PULL                          ; restore registers
```

```
  RETFIE                        ; return from interrupt

; ********************************************************************
;  RX_INT_HANDLER - handles the received commands on serial com
;  called from inside the interrupt handler
RX_INT_HANDLER
  BTFSS     RCSTA,OERR          ; test for overrun error
  GOTO      RX_CHECK_FRAMING
  ; when overrun, uart will stop receiving the continuous stream
  ; receive bit must then be re-set
  BCF       RCSTA,CREN          ; clear continuous receive bit
  BSF       RCSTA,CREN          ; and set it again
RX_CHECK_FRAMING
  BTFSS     RCSTA,FERR          ; check from framing errors
  GOTO      RX_CHECK_BUFFER
  ; framing error do not store this byte
  ; read rx reg and discard byte
  GOTO      RX_DISCARD_BYTE
RX_CHECK_BUFFER
  MOVF      RX_Buffer_Count,W   ; test if empty
  XORLW     RX_BUFFER_SIZE
  BTFSC     STATUS,Z
  GOTO      RX_BUFFER_FULL
  MOVF      RX_Buffer_InPtr,W   ; get address for indirect addressing
  MOVWF     FSR                 ; setup fsr
  MOVF      RCREG,W             ; get received byte
  MOVWF     INDF                ; and store it in buffer
  INCF      RX_Buffer_Count,F   ; inc buffer counter
  ; update pointers
  INC_BUFFER  RX_Buffer_InPtr,RX_Buffer,RX_BUFFER_SIZE
  RETURN
RX_BUFFER_FULL
  BSF       _BufferOverrun      ; no rom for more bytes, set overrun flag
RX_DISCARD_BYTE                 ; and clear the last byte (no room to store it)
  ; optional an error flag could be set to indicate comm error.
  MOVF      RCREG,W             ; read byte and discard
  RETURN

; ********************************************************************
;  TX_INT_HANDLER - handles the transmission of bytes on serial com
;  called from inside the interrupt handler
TX_INT_HANDLER
  MOVF      TX_Buffer_Count,W   ; get number of bytes
  BTFSC     STATUS,Z            ; buffer empty ?
  GOTO      TX_BUFFER_EMPTY
  MOVF      TX_Buffer_OutPtr,W  ; get address for indirect addressing
  MOVWF     FSR                 ; setup fsr
  MOVF      INDF,W              ; get byte
  MOVWF     TXREG               ; and put it in tx reg
  DECF      TX_Buffer_Count,F   ; decrement buffer counter
  ; update pointers
  INC_BUFFER  TX_Buffer_OutPtr,TX_Buffer,TX_BUFFER_SIZE
  RETURN
TX_BUFFER_EMPTY
  ; no more bytes to send disable TX irq
  ; code is to avoid bank switching ( using FSR )
  MOVLW     PIE1                ; get address for tx irq enable
  MOVWF     FSR                 ; setup fsr
  BCF       INDF,TXIE           ; and disable tx irq
  RETURN

; ********************************************************************
;  TX_ADD_BUFFER - Puts one byte in serial tx buffer,  blocks until room available
;  Make sure to be at bank0 ( buffer pointers in bank0, buffers in bank1 )
TX_ADD_BUFFER
  MOVWF     TX_Temp             ; store byte temporarily
TX_ADD_TST
  MOVF      TX_Buffer_Count,W   ; get count
  XORLW     TX_BUFFER_SIZE
  BTFSC     STATUS,Z            ; test if any room
  GOTO      TX_ADD_TST          ; no room, wait until there is
  MOVF      TX_Buffer_InPtr,W   ; get address to store byte
  MOVWF     FSR                 ; setup fsr
  MOVF      TX_Temp,W           ; get byte
  MOVWF     INDF                ; and store it
  ; update buffer pointers
  INC_BUFFER  TX_Buffer_InPtr,TX_Buffer,TX_BUFFER_SIZE
  INCF      TX_Buffer_Count,F   ; increment byte count
        ; MUST not be done until after byte is stored in buffer
        ; and pointers updated
  MOVLW     PIE1                ; get address for peripheral irq
  MOVWF     FSR                 ; setup fsr
  BSF       INDF,TXIE           ; and enable tx irq
  RETURN

; ********************************************************************
;  RX_GET_BUFFER - Gets one byte from in serial rx buffer, if available
;  If no bytes in buffer zero flag is set else zero flag is cleared
;  Make sure to be at bank0 ( buffer pointers in bank0, buffers in bank1 )
RX_GET_BUFFER
  MOVF      RX_Buffer_Count,F   ; check if anything available ?
  BTFSC     STATUS,Z            ;
   RETURN                       ; nope, nothing there, NOTE zero flag set
  MOVF      RX_Buffer_OutPtr,W  ; get pointer
  MOVWF     FSR                 ; setup FSR
  ; update buffer pointers
  INC_BUFFER  RX_Buffer_OutPtr,RX_Buffer,RX_BUFFER_SIZE
  MOVF      INDF,W              ; get byte to W
  DECF      RX_Buffer_Count,F   ; decrement counter
  BCF       STATUS,Z            ; make sure zero flag is clear
   RETURN
```

```
; ********************************************************************
; INIT - Cold start vector, called at startup
; initialize all ports to known state before setup routines are called
INIT
  ; pclath and status is already cleared !
  ; before entering this init routine
  CLRF      INTCON              ; ensure int reg is clear
  CLRF      PIR1                ; clear peripheral irqs
  CLRF      PIR2                ; clear peripheral irqs
  ; make sure all individual irq's are disabled
  MOVLW     PIE1                ; get address for peripheral irq enable
  MOVWF     FSR                 ; setup fsr
  CLRF      INDF                ; and clear irq enable flags
  MOVLW     PIE2                ; get address for second peripheral irq enable
  MOVWF     FSR                 ; setup fsr
  CLRF      INDF                ; and clear irq enable flags
  CLRF      PORTA               ;
  CLRF      PORTB               ; clear output data latches
  CLRF      PORTC               ;
                  ; Port D and E?

  ; call initialization routines for peripherals/ports
  ; note must be at bank0 during init

  ; NOTE ! DO NOT CHANGE ORDER OF THESE ROUTINES !!

  ; clear all user ram ( set to all 0's )
  CALL      CLEAR_RAM
  ; setup our ports to in/out/analogue/rx/tx/etc
  ; must be done before calling any other INIT_XXX routine
  ; as most of them depends on pin settings
  CALL      INIT_PORTS
  CALL      INIT_UART           ; setup uart
  CALL      INIT_BUFFERS        ; setup uart buffers
  CALL      INIT_VARS           ; initialize variables that require it
  ENABLE_IRQ                    ; all pins/peripherals configured, enable GIE
;  All configured and ready to rumble

;Issue reset command to BT and wait for confirmation of reset complete
DOWHILE1                        ;do{
WAIT1
  MOVF      NumHCICommandPackets,F  ; check how many packets BT module has room
                            ;for atm
  BTFSC     STATUS,Z
  GOTO      WAIT1                 ; goto wait1 if NumHCICommandPackets = 0

  MOVLW     HCI_Reset             ; BT should be ready for a command, so:
  CALL      SEND_STRING           ; send reset command to BT module
  DECF      NumHCICommandPackets,F  ; NumHCICommandPackets--

  CALL      Wait_HCI_Reset_Complete ; Status=Wait_HCI_Reset(); await a status of
                            ;ok

  MOVF      RequestStatus,F     ;} while (Status != 0);  - if not ok, re-reset
  BTFSC     STATUS,Z
   GOTO      DOWHILE1

;Issue ReadBuffer command to BT and wait for confirmation of ReadBuffer complete
DOWHILE2                        ;do{
WAIT2
  MOVF      NumHCICommandPackets,F  ; check how many packets BT module has room
                            ;for atm
  BTFSC     STATUS,Z
  GOTO      WAIT2                 ; goto wait2 if NumHCICommandPackets = 0

  MOVLW     HCI_Read_Buffer_Size    ; BT should be ready for a command, so:
  CALL      SEND_STRING           ; send Read Buffer Size command to BT module
  DECF      NumHCICommandPackets,F  ; NumHCICommandPackets--

  CALL      Wait_HCI_Read_Buffer_Size ; Status=Wait_HCI_Read_Buffer_Size(); await a status of ok

  MOVF      RequestStatus,F     ;} while (Status != 0);  - if not ok, re-reset
  BTFSC     STATUS,Z
   GOTO      DOWHILE2

;Issue ReadBDAddr command to BT and wait for confirmation of ReadBDAddr complete
DOWHILE3                        ;do{
WAIT3
  MOVF      NumHCICommandPackets,F  ; check how many packets BT module has room for atm
  BTFSC     STATUS,Z
  GOTO      WAIT3                 ; goto wait3 if NumHCICommandPackets = 0

  MOVLW     HCI_Read_BD_ADDR        ; BT should be ready for a command, so:
  CALL      SEND_STRING           ; send Read BD Addr command to BT module
  DECF      NumHCICommandPackets,F  ; NumHCICommandPackets--

  CALL      Wait_HCI_Read_BD_ADDR   ; Status=Wait_HCI_Read_BD_Addr(); await a status of ok

  MOVF      RequestStatus,F     ;} while (Status != 0);  - if not ok, re-reset
  BTFSC     STATUS,Z
   GOTO      DOWHILE3

;  CLRF      RequestStatus       ;
;  INCF      RequestStatus       ; Status = 1
;Issue CreateConnection command to BT and wait for HCI_Connection_Complete
DOWHILE4                        ; do{ do{
WAIT4
  MOVF      NumHCICommandPackets,F  ; check how many packets BT module has room for atm
  BTFSC     STATUS,Z
  GOTO      WAIT4                 ; goto wait4 if NumHCICommandPackets = 0

  MOVLW     HCI_Create_Connection   ; BT should be ready for a command, so:
```

```
  CALL      SEND_STRING            ; send Read BD Addr command to BT module
  DECF      NumHCICommandPackets,F  ; NumHCICommandPackets--
  CALL      Wait_HCI_Create_Conn_Status; Status=Wait_HCI_Create_Connection_Status()
  MOVF      RequestStatus,F     ;} while (Status != 0);  - if not ok, re-reset
  BTFSC     STATUS,Z
   GOTO      DOWHILE4
  CALL      Wait_HCI_Conn_Complete  ; Status=Wait_HCI_Connection_Complete()
  MOVF      RequestStatus,F     ;} while (Status != 0);  - if not ok, re-reset
  BTFSC     STATUS,Z
   GOTO      DOWHILE4

;At this point, we should have a live connection to a remote BT device!
;    ACLDataPacket[1] = 0x02;           /* Transport layer header for this ACL data packet */
;    ACLDataPacket[2] = Conn_Handle[1];  /* LSB of the ConnectionHandle to address this packet to */
;    ACLDataPacket[3] = Conn_Handle[0];  /* MSB of the Connection Handle, and PB (bit 6), BC (bit 7) ;
                                           flags */
;    ACLDataPacket[4] = 0x12;           /* Data payload length in bytes, LSB */
;    ACLDataPacket[5] = 0x00;           /* Data payload length in bytes, MSB */

   ;From Init code up to this point ....
   ;ADCON0 is set up to be:  Fosc/8 Hz (), reading AN0, ADOFF
    BSF     ADCON0,ADON         ;  Power up AD module - how long does it need to power up?
   ;ADCON1 is set up to be:  8 MSbits of ADC in ADRESH, AN7..0 = analog in, VHref = Vdd, VLref = Vss
MAIN_LOOP
  ; main loop
  ;Address VisionCell 1 (Index 0x00)
   ; address phototransistors to sample through port B (B2:B1)
    BCF       PORTB,2             ; Address bank 0 of analog pins through demuxes
    BCF       PORTB,1             ;
   ;wait for addressing to take effect
   ;/* At crystal speed of 3.6864 MHz, an instr. cycle is ~1.085 microseconds */
   ;/* Spec sheet for Demux (CD4052BC) lists maximum time for address to take effect is 1 microsecond */
   ;/* will throw in an extra nop for good measure, will need more if oscillator is faster */
    ;NOP  maybe don't need these as selecting the CHS bits takes time
    ;NOP
  ; select AN4 (Eye0) line as input for analog
    BSF     ADCON0,CHS2
    BCF     ADCON0,CHS1
    BCF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop0
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop0            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray         ; Store in VisionArray[0]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN5 (Eye1) line as input for analog
    BSF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop1
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop1            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+1       ; Store in VisionArray[1]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN7 (Eye3) line as input for analog
    BSF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop3
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop3            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+3       ; Store in VisionArray[3]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN6 (Eye2) line as input for analog
    BCF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop2
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop2            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+2       ; Store in VisionArray[2]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

   ; address phototransistors to sample through port B (B2:B1)
    BSF       PORTB,1             ; Address bank 1 of analog pins through demuxes
   ;wait for addressing to take effect
    NOP
    NOP
```

```
  ; select AN4 (Eye4) line as input for analog
    BCF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop4
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop4            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+4         ; Store in VisionArray[4]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN5 (Eye5) line as input for analog
    BSF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop5
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop5            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+5       ; Store in VisionArray[5]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN7 (Eye7) line as input for analog
    BSF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop7
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop7            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+7       ; Store in VisionArray[7]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN6 (Eye6) line as input for analog
    BCF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop6
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop6            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+6       ; Store in VisionArray[6]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

   ; address phototransistors to sample through port B (B2:B1)
    BSF       PORTB,2             ; Address bank 2 of analog pins through demuxes
    BCF       PORTB,1             ;
   ;wait for addressing to take effect
    NOP
    NOP
  ; select AN4 (Eye8) line as input for analog
    BCF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop8
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop8            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+8       ; Store in VisionArray[8]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN5 (Eye9) line as input for analog
    BSF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop9
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop9            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+9       ; Store in VisionArray[9]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN7 (Eye11) line as input for analog
    BSF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
```

```
ADCLoop11
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop11           ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+11       ; Store in VisionArray[11]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN6 (Eye10) line as input for analog
    BCF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop10
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop10            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+10       ; Store in VisionArray[10]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

   ; address phototransistors to sample through port B (B2:B1)
    BSF       PORTB,1             ; Address bank 3 of analog pins through demuxes
   ;wait for addressing to take effect
    NOP
    NOP
  ; select AN4 (Eye0) line as input for analog
    BCF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop12
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop12           ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+12         ; Store in VisionArray[12]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN5 (Eye13) line as input for analog
    BSF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop13
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop13           ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+13       ; Store in VisionArray[13]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN7 (Eye15) line as input for analog
    BSF     ADCON0,CHS1
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop15
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop15           ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+15       ; Store in VisionArray[15]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK

  ; select AN6 (Eye14) line as input for analog
    BCF     ADCON0,CHS0
  ;wait Tacq s for Capacitor to charge fully (sample time)
   ;several nops here, 20?
  ; start AD conversion from capacitor
    BSF     ADCON0,GO           ;Set GO/'DONE bit to start conversion
ADCLoop14
    BTFSC   ADCON0,NOT_DONE     ; Check if conversion is done
     GOTO    ADCLoop14            ; A/D still converting, so wait
    MOVF    ADRESH,W            ; Read the A/D high order byte
    ;store value in VisionArray
    BSF     STATUS,RP1          ;SET TO RAM BANK 2
    MOVWF   VisionArray+14       ; Store in VisionArray[14]
    BCF     STATUS,RP1          ;BACK TO RAM BANK 0 RIGHT QUICK
;*************** EYE DATA IS DONE ***********************************

  ;Acquire SensorStates

  ;Acquire MotorStates

  ;Send VisionArray to BT

  ;Send SensorStates to BT

  ;Send MotorStates to BT

;  GOTO     MAIN_LOOP
```

```
  ; test for specific events
MAIN_COM_EVENT
  ; test if a character is received ( i.e. saved in rx buffer )
  CALL      RX_GET_BUFFER
  BTFSC     STATUS,Z            ; test zero flag
  GOTO      MAIN_DONE           ; zero flag set=no byte received, test next event

  ; ****************************
  ; a byte is received !
  ; ****************************
  ; do whatever needs to be done
  ; ****************************
  ; Something that's neat to put here is the code
  ; generated by Nicolai's autogenerated parser ( top stuff )
  ; from the picfaq site:
  ; http://www.piclist.com/techref/default.asp?url=piclist/codegen
  ; ****************************
MAIN_DONE
  NOP
  NOP
  ; and return to main loop
  GOTO      MAIN_LOOP


; *********************************************************************
;  CLEAR_RAM - Reset all general purpose ram to 0's
;  Note ! does not clear watchdog, add CLRWDT where appropiate if enabled
;  Make sure to be at bank0
CLEAR_RAM
  MOVLW     0x20                ; start ram bank0
  MOVWF     FSR
CLEAR_BANK0
  CLRF      INDF                ; Clear a register pointed to be FSR
  INCF      FSR,F
  MOVLW     0x7F                ; Test if at top of memory bank0
  SUBWF     FSR,W
  BNZ       CLEAR_BANK0         ; Loop until all cleared
  MOVLW     0xA0                ; start ram bank1
  MOVWF     FSR
CLEAR_BANK1
  CLRF      INDF                ; Clear a register pointed to be FSR
  INCF      FSR,F
  ; note this could also be set to 0xFF or 0xEF as the top 16 bytes are mirrored from
  ; bank0
  MOVLW     0xEF                ; Test if at top of memory bank1
  SUBWF     FSR,W
  BNZ       CLEAR_BANK1         ; Loop until all cleared
  BANK2                         ; select bank2/3 ( with indirect addressing )
  MOVLW     0x10                ; start ram bank2
  MOVWF     FSR
CLEAR_BANK2
  CLRF      INDF                ; Clear a register pointed to be FSR
  INCF      FSR,F
  ; note this could also be set to 0x7F or 0x70 as the top 16 bytes are mirrored from
  ; bank0
  MOVLW     0x70                ; Test if at top of memory bank2
  SUBWF     FSR,W
  BNZ       CLEAR_BANK2         ; Loop until all cleared
  MOVLW     0x90                ; start ram bank3
  MOVWF     FSR
CLEAR_BANK3
  CLRF      INDF                ; Clear a register pointed to be FSR
  INCF      FSR,F
  ; note this could also be set to 0xFF or 0xEF as the top 16 bytes are mirrored from
  ; bank0
  MOVLW     0xEF                ; Test if at top of memory bank3
  SUBWF     FSR,W
  BNZ       CLEAR_BANK3         ; Loop until all cleared
  BANK0                         ; set back to bank0
  RETURN

; *********************************************************************
;
;  INIT_PORTS - Initializes all ports on the PIC
;  i.e sets the pins as in/out/analog/etc
;  Make sure to be at bank0
INIT_PORTS
  BSF       STATUS,RP0          ;Select memory page 1
  CLRF      ADCON1              ;Set all PORTA,E pins to A/D inputs, VREF to Vdd (R)
  ; set in/out for porta pins
  MOVLW     0xFF
  MOVWF     TRISD               ;set Port D to input from vehicle (R)
  MOVWF     TRISA               ;set Port A to input - from Eye, vehicle (R)
  MOVLW     B'00000111'         ;L=0x07
  MOVWF     TRISE               ;set Port E to input - from Eye (R)
  MOVLW     B'11111001'         ;L=0xF9 ... TRISB = FF at (Reset)
  MOVWF     TRISB               ;set Port B to input - except B1,2 to Eye Addr
  ; setup PORTC
  ; note PORTC must be setup properly when using SPI/UART/CCP/TIMER
  ; look in data sheet, some setups are 'illogical' and/or overridden
  ; as TX pin configured as input etc.
  ; set in/out for portc pins
  MOVLW     b'11010000'
        ; 7-6 for uart must be set, 4(SDI) MUST be input i.e set (for SPI master)
        ;  5 ( SDO ) must be cleared, 3 (SCK) must be cleared
        ; for master mode. 1-2 is for CCP module, 0 is for timer inp.
  MOVWF     TRISC               ;set Port C control register
  ; setup OPTION reg
;  MOVLW     OPTION_REG          ; get address for option reg
;  MOVWF     FSR                 ; setup fsr
;  MOVLW     b'00000000' ; pull up portb by latch, int edge falling,TMR0 source internal
        ; TMR0 source edge inc on low->high, prescaler to Timer0, TMR0 rate 1:2
```

```
;  MOVWF     INDF  ; and set it
  BCF       STATUS,RP0          ;Select memory page 0
  ; enable/shutoff ad/module - RR, try not enabling AD yet for power consumption
  MOVLW     (1<<ADCS0);|(1<<ADON); enable ad-module, ad clock is osc/8
  MOVWF     ADCON0              ; and set it
  RETURN

; ********************************************************************
;
;  INIT_UART - Initializes UART
;  enables receiver and transmitter
;  Make sure to be at bank0
INIT_UART
  ; make sure pins are setup before calling this routine
  ; TRISC:6 and TRISC:7 must be set ( as for output, but operates as input/output )
  ; furthermore its advised that interrupts are disabled during this routine
  ; setup baudrate
  MOVLW     SPBRG               ; get address for serial baud reg
  MOVWF     FSR                 ; setup fsr
  MOVLW     CALC_LOW_BAUD(57600); calculate baudrate 57600 with brgh=0
  MOVWF     INDF                ; and store it
  ; enable transmitter
  MOVLW     TXSTA               ; get address for serial enable reg
  MOVWF     FSR                 ; setup fsr
  MOVLW     (1<<TXEN)           ;|(1<<BRGH) preset enable transmitter and low speed mode
  MOVWF     INDF                ; and set it
  ; enable receiver
  MOVLW     (1<<SPEN)|(1<<CREN) ; preset serial port enable and continuous receive
  MOVWF     RCSTA               ; set it
  ; enable receiver interrupt
  MOVLW     PIE1                ; get address for peripheral irq's
  MOVWF     FSR                 ; setup fsr
  BSF       INDF,RCIE           ; enable receiver irq
  BSF       INTCON,PEIE         ; and peripheral irq must also be enabled
  RETURN

; ********************************************************************
;
;  INIT_BUFFERS - Initializes all buffers and pointers
;  Make sure to be at bank0
INIT_BUFFERS
  ; setup receive buffer
  CLRF      RX_Buffer_Count     ; clear counter
  MOVLW     RX_Buffer           ; get base address for buffer
  MOVWF     RX_Buffer_InPtr     ; save as in address
  MOVWF     RX_Buffer_OutPtr    ; and out address
  ; setup transmit buffer
  CLRF      TX_Buffer_Count     ; clear counter
  MOVLW     TX_Buffer           ; get base address for buffer
  MOVWF     TX_Buffer_InPtr     ; save as in address
  MOVWF     TX_Buffer_OutPtr    ; and out address
  ;setup ACL Data Packet buffer
;  MOVLW    ACLDataPacket       ; get base address for buffer
;  MOVWF    ACLDataPtr          ; save as in address
  RETURN

; ********************************************************************
;
;  INIT_VARS - Initialises variables requiring a non-zero value
;  Make sure to be at bank0
INIT_VARS
  BSF       STATUS,RP1          ;Select memory bank 2
  MOVLW     1
  MOVWF     NumHCICommandPackets;=1! From ROK101008, thought it might be higher
  MOVLW     1
  MOVWF     RequestStatus       ; = 1
  MOVLW     0xFF
  MOVWF     BD_ADDR_remote_1    ; = 0xFF
  MOVLW     0xEE
  MOVWF     BD_ADDR_remote_2    ; = 0xEE
  MOVLW     0xDD
  MOVWF     BD_ADDR_remote_3    ; = 0xDD
  MOVLW     0xCC
  MOVWF     BD_ADDR_remote_4    ; = 0xCC
  MOVLW     0xBB
  MOVWF     BD_ADDR_remote_5    ; = 0xBB
  MOVLW     0xAA
  MOVWF     BD_ADDR_remote_6    ; = 0xAA
  BCF       STATUS,RP1          ;Select memory bank 0
  RETURN


;***********************************************
; Begin RECEIVE_HCI_PACKET
;***********************************************
RECEIVE_HCI_PACKET
;char Receive_HCI_Packet(char *PacketPT){
;We will read bytes from the RxBuffer in the form of packets and save them in PacketBuff
; It is expected that the first byte in the FIFO is the transport header, packet type
WAIT_EVENT1
  CALL      RX_GET_BUFFER       ; store character in w if one is waiting
  BTFSC     STATUS,Z            ; test zero flag
   GOTO      WAIT_EVENT1         ; zero flag set = no byte received, wait more
  MOVWF     Packet_Type         ; Type = InChar();
  ADDLW     -EVENT              ; Type == Event?
  BTFSS     STATUS,Z            ;
   GOTO      DataTest            ; check if we got a data packet
                                 ; Packet is an event, add information to event buffer
WAIT_EVENT2
  CALL      RX_GET_BUFFER       ; Take connection handle LSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
   GOTO      WAIT_EVENT2         ; zero flag set = no byte received, wait more
  MOVWF     PacketBuff          ;  EventCode = InChar();
```

```
WAIT_EVENT3
  CALL      RX_GET_BUFFER       ; Take connection handle LSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
   GOTO      WAIT_EVENT3         ; zero flag set = no byte received, wait more
  MOVWF     PacketBuff+1        ;  ParamLength = InChar();
 ;/* Following possibly loses data, but safer than allowing array overrun */
  ADDLW     -MAXEVENTSIZE-2     ; if (ParamLength > (MAXEVENTSIZE-2))
  BTFSS     STATUS,C            ; then skip the next line
   GOTO      EndLengthTest
  MOVLW     MAXEVENTSIZE-2      ;*(PacketPT + 1) = MAXEVENTSIZE-2
  MOVWF     PacketBuff+1        ;
EndLengthTest

    ;InEventBitString(&ParamData, ParamLength)
  CLRF      LoopI               ; for (i=1; i<=length; i++) {
  INCF      LoopI,F
  GOTO      LoopTest4
WAIT_EVENT4
  INCF        LoopI,W             ; W = LoopI + 1
  ADDLW       PacketBuff          ; W = W + &PacketBuff
  MOVWF       FSR                 ; FSR = W
  BCF           STATUS,IRP          ;   CLR IRP, ?????
  CALL      RX_GET_BUFFER       ; Take event char out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
   GOTO     WAIT_EVENT4         ; zero flag set = no byte received, wait more
  MOVWF     INDF                ;save the char in PacketBuff+1+i

  INCF      LoopI,F             ; }
LoopTest4
  MOVF      LoopI,W
  SUBWF     PacketBuff+1,W
  BTFSC     STATUS,C
   GOTO     WAIT_EVENT4
  RETURN                        ; Return Type, Event packet has been buffered

;  } else {                                 /* Packet is of an unrecognized type, or error */
DataTest
  ADDLW     EVENT-ACLData       ; Type == Data?
  BTFSS     STATUS,Z            ;
   RETURN                       ; End Receive_packet, we don't handle types other than event&data at this time
                                ;       Expect calling code to handle errors like this.
WAIT_DATA1
  CALL      RX_GET_BUFFER       ; Take connection handle LSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
  GOTO      WAIT_DATA1          ; zero flag set = no byte received, wait more
WAIT_DATA2
  CALL      RX_GET_BUFFER       ; Take connection handle MSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
  GOTO      WAIT_DATA2          ; zero flag set = no byte received, wait more
WAIT_DATA3
  CALL      RX_GET_BUFFER       ; Take length LSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
  GOTO      WAIT_DATA4          ; zero flag set = no byte received, wait more
  MOVWF     DataLength          ;
WAIT_DATA4
  CALL      RX_GET_BUFFER       ; Take length MSB out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
  GOTO      WAIT_DATA4          ; zero flag set = no byte received, wait more
  MOVWF     DataLength+1        ;  CHECK FOR ACCURACY OF LSB:MSB assumption

  CLRF      LoopI               ; for (i=1; i<=length; i++) {
  INCF      LoopI,F
  CLRF      LoopI+1
  GOTO      LoopTest5
WAIT_DATA5
  CALL      RX_GET_BUFFER       ; Take dummy char out of the FIFO
  BTFSC     STATUS,Z            ; test zero flag
   GOTO     WAIT_DATA5          ; zero flag set = no byte received, wait more
  INCF      LoopI,F             ; }
  BTFSC     STATUS,Z
   INCF     LoopI+1,F
LoopTest5
  MOVF      DataLength+1,W
  XORLW     128                 ; CHECK FOR meaning of 128
  MOVWF     btemp
  MOVF      LoopI+1,W
  XORLW     128
  SUBWF     btemp,W
  BTFSS     3,2
   GOTO     LoopEnd5
  MOVF      LoopI,W
  SUBWF     DataLength,W
LoopEnd5
  BTFSC     3,0
   GOTO     WAIT_DATA5
  RETURN                        ; Data packet was removed from buffer
;***********************************************
; End RECEIVE_HCI_PACKET
;***********************************************


;/*----------------------Wait_HCI_Reset_Complete----------------------
;  Wait for the HCI Command Complete event with the HCI_Reset command opcode
;  to arrive in the RxFifo and be parsed - check if status is ok
;  Note: enters persistent loop until desired event arrives.  May want time-out feature?
;  Inputs: none
;  Outputs: status of command (0x00 = success, error code = failure) */
;char Wait_HCI_Reset(void){
Wait_HCI_Reset_Complete         ;  while (1){
  CALL      RECEIVE_HCI_PACKET  ;  Blocking wait/receive: doesn't return until a packet is found
```

```
  MOVF      Packet_Type,W
  SUBLW     EVENT               ;if (TypeObtained == EVENT) {
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Reset_Complete;        /* because we are waiting for a reset, ignore all other ;packets
  MOVF      PacketBuff,W        ;if (PacketObtained[0] == 0x0E){
  SUBLW     0x0E                ;
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Reset_Complete;        /* because we are waiting for a reset, ignore all other ;events
  MOVF      PacketBuff+2,W      ;Num_HCI_Command_Packets = PacketObtained[2]
  MOVWF     NumHCICommandPackets;
  MOVF      PacketBuff+3,W      ;if ((PacketObtained[3]==0x03) &&
  SUBLW     0x03
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Reset_Complete
  MOVF      PacketBuff+4,W      ; (PacketObtained[4]==0x0C)) { //Event is responding to an HCI_Reset
  SUBLW     0x0C
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Reset_Complete
  MOVF      PacketBuff+5,W      ; return PacketObtained[5];
  MOVWF     RequestStatus
  RETURN                        ; Optionally, insert different handling of 'wrong' packets
;END Wait_HCI_Reset_Complete


;/*----------------------Wait_HCI_Read_Buffer_Size--------------------
;  Wait for the HCI Command Complete event with the HCI_Read_Buffer_Size command opcode
;  to arrive in the RxFifo and be parsed - check if status is ok
;  Note: enters persistent loop until desired event arrives.  May want time-out feature?
;  Inputs: none
;  Outputs: status of command (0x00 = success, error code = failure) */
;char Wait_HCI_Reset(void){
Wait_HCI_Read_Buffer_Size       ;  while (1){
  CALL      RECEIVE_HCI_PACKET  ;  Blocking wait/receive: doesn't return until a packet is found
  MOVF      Packet_Type,W
  SUBLW     EVENT               ;if (TypeObtained == EVENT) {
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Reset_Complete;        /* because we are waiting for a reset, ignore all other
                                                        ;packets
  MOVF      PacketBuff,W        ;if (PacketObtained[0] == 0x0E){
  SUBLW     0x0E                ;
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_Buffer_Size; because we are waiting for a reset, ignore all other events
  MOVF      PacketBuff+2,W      ;Num_HCI_Command_Packets = PacketObtained[2]
  MOVWF     NumHCICommandPackets;
  MOVF      PacketBuff+3,W      ;if ((PacketObtained[3]==0x05) &&
  SUBLW     0x05
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_Buffer_Size
  MOVF      PacketBuff+4,W      ; (PacketObtained[4]==0x10)) { //Event is responding to an HCI_Read_Buffer_Size
  SUBLW     0x10
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_Buffer_Size
  MOVF      PacketBuff+5,W      ; return PacketObtained[5];
  MOVWF     RequestStatus
  MOVF      PacketBuff+6,W      ;HC_ACL_Data_Packet_Length = ((int)PacketObtained[7]<<8) + PacketObtained[6];
/* 2 bytes */
  MOVWF     HC_ACL_Data_Packet_LengthL
  MOVF      PacketBuff+7,W      ;
  MOVWF     HC_ACL_Data_Packet_LengthH
  MOVF      PacketBuff+8,W      ;HC_SCO_Data_Packet_Length = PacketObtained[8];
  MOVWF     HC_SCO_Data_Packet_Length
  MOVF      PacketBuff+9,W      ;HC_Total_Num_ACL_Data_Packets = ((int)PacketObtained[10]<<8) + PacketObtained[9];
/* 2 bytes */
  MOVWF     HC_Total_Num_ACL_Data_PacketsL
  MOVF      PacketBuff+10,W      ;
  MOVWF     HC_Total_Num_ACL_Data_PacketsH
  MOVF      PacketBuff+11,W      ;HC_Total_Num_SCO_Data_Packets = ((int)PacketObtained[12]<<8) +
PacketObtained[11];  /* 2 bytes */
  MOVWF     HC_Total_Num_SCO_Data_PacketsL
  MOVF      PacketBuff+12,W      ;
  MOVWF     HC_Total_Num_SCO_Data_PacketsH
  RETURN                        ; Optionally, insert different handling of 'wrong' packets
;END Wait_HCI_Read_Buffer_Size


;/*----------------------Wait_HCI_Read_BD_ADDR--------------------------------------
;  Wait for the HCI Command Complete event with the HCI_Read_BD_ADDR command opcode
;  to arrive in the RxFifo and be parsed - check if status is ok and save BD_ADDR string globally
;  Note: enters persistent loop until desired event arrives.  May want time-out feature?
;  Inputs: none
;  Outputs: status of command (0x00 = success, error code = failure) */
;char Wait_HCI_Read_BD_ADDR(void){
Wait_HCI_Read_BD_ADDR           ;  while (1){
  CALL      RECEIVE_HCI_PACKET  ;  Blocking wait/receive: doesn't return until a packet is found
  MOVF      Packet_Type,W
  SUBLW     EVENT               ;if (TypeObtained == EVENT) {
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_BD_ADDR ;
  MOVF      PacketBuff,W        ;if (PacketObtained[0] == 0x0E){
  SUBLW     0x0E                ;
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_BD_ADDR ;
  MOVF      PacketBuff+2,W      ;Num_HCI_Command_Packets = PacketObtained[2]
  MOVWF     NumHCICommandPackets;
  MOVF      PacketBuff+3,W      ;if ((PacketObtained[3]==0x09) &&
  SUBLW     0x09
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_BD_ADDR
  MOVF      PacketBuff+4,W      ; (PacketObtained[4]==0x10)) { //Event is responding to an HCI_Read_BD_ADDR
  SUBLW     0x10
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Read_BD_ADDR
```

```
;  MOVF      PacketBuff+6,W      ; BD_ADDR_local[5] = PacketObtained[6];
;  MOVWF     BD_ADDR_local+5     ;
;  MOVF      PacketBuff+7,W      ; BD_ADDR_local[4] = PacketObtained[7];
;  MOVWF     BD_ADDR_local+4     ;
;  MOVF      PacketBuff+8,W      ;BD_ADDR_local[3] = PacketObtained[8];
;  MOVWF     BD_ADDR_local+3     ;
;  MOVF      PacketBuff+9,W      ;BD_ADDR_local[2] = PacketObtained[9];
;  MOVWF     BD_ADDR_local+2     ;
;  MOVF      PacketBuff+10,W     ;BD_ADDR_local[1] = PacketObtained[10];
;  MOVWF     BD_ADDR_local+1     ;
;  MOVF      PacketBuff+11,W     ;BD_ADDR_local[0] = PacketObtained[11];
;  MOVWF     BD_ADDR_local       ;
  MOVF      PacketBuff+5,W      ; return PacketObtained[5];
  MOVWF     RequestStatus
  RETURN                        ; Optionally, insert different handling of 'wrong' packets
;END Wait_HCI_Read_BD_ADDR


;/*----------------------Wait_HCI_Create_Conn_Status----------------------
;  Wait for the HCI Command Complete event with the HCI_Reset command opcode
;  to arrive in the RxFifo and be parsed - check if status is ok
;  Note: enters persistent loop until desired event arrives.  May want time-out feature?
;  Inputs: none
;  Outputs: status of command (0x00 = success, error code = failure) */
;char Wait_HCI_Create_Connection_Status(void){
Wait_HCI_Create_Conn_Status     ;  while (1){
  CALL      RECEIVE_HCI_PACKET  ;  Blocking wait/receive: doesn't return until a packet is found
  MOVF      Packet_Type,W
  SUBLW     EVENT               ;if (TypeObtained == EVENT) {
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Create_Conn_Status;
  MOVF      PacketBuff,W        ;if (PacketObtained[0] == 0x0F){
  SUBLW     0x0F                ;
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Create_Conn_Status;
  MOVF      PacketBuff+3,W      ;Num_HCI_Command_Packets = PacketObtained[3]
  MOVWF     NumHCICommandPackets;
  MOVF      PacketBuff+4,W      ;if ((PacketObtained[4]==0x05) &&
  SUBLW     0x05
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Create_Conn_Status
  MOVF      PacketBuff+5,W      ; (PacketObtained[5]==0x04)) { //Event is responding to an HCI_Create_Conn
  SUBLW     0x04
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Create_Conn_Status
  MOVF      PacketBuff+2,W      ; return PacketObtained[2];
  MOVWF     RequestStatus
  RETURN                        ; Optionally, insert different handling of 'wrong' packets
;END Wait_HCI_Create_Conn_Status


;/*----------------------Wait_HCI_Conn_Complete----------------------------------------
;  Wait for the HCI Connection Complete event to arrive in the RxFifo and be parsed.
;  Check if status is ok and save BD_ADDR string globally
;  Note: enters persistent loop until desired event arrives.  May want time-out feature?
;  Inputs: none
;  Outputs: status of command (0x00 = success, error code = failure) */
;char Wait_HCI_Connection_Complete(void){
Wait_HCI_Conn_Complete          ;  while (1){
  CALL      RECEIVE_HCI_PACKET  ;  Blocking wait/receive: doesn't return until a packet is found
  MOVF      Packet_Type,W
  SUBLW     EVENT               ;if (TypeObtained == EVENT) {
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Conn_Complete ;
  MOVF      PacketBuff,W        ;if (PacketObtained[0] == 0x03){
  SUBLW     0x03                ;
  BTFSS     STATUS,Z
   GOTO      Wait_HCI_Conn_Complete ;
  MOVF      PacketBuff+3,W      ;Conn_HandleL = PacketObtained[3];
  MOVWF     Conn_HandleL        ;
  MOVF      PacketBuff+4,W      ;Conn_HandleH = PacketObtained[4];
  MOVWF     Conn_HandleH        ;
;        /*BD_ADDR[5] = PacketObtained[5];
;        BD_ADDR[4] = PacketObtained[6];
;        BD_ADDR[3] = PacketObtained[7];
;        BD_ADDR[2] = PacketObtained[8];
;        BD_ADDR[1] = PacketObtained[9];
;        BD_ADDR[0] = PacketObtained[10];*/  /* Wait to see if we really need to look at this BD_ADDR value
(1/04/2002) */
  MOVF      PacketBuff+11,W     ;Link_Type = PacketObtained[11];
  MOVWF     Link_Type           ;
  MOVF      PacketBuff+12,W     ;Encryption_Mode = PacketObtained[12];
  MOVWF     Encryption_Mode     ;
  MOVF      PacketBuff+2,W      ; return PacketObtained[2];
  MOVWF     RequestStatus
  RETURN                        ; Optionally, insert different handling of 'wrong' packets
;END Wait_HCI_Conn_Complete


; ********************************************************************
; SEND_STRING - Send a string on the serial communication ( put into tx buffer )
;  On entry, W contains the string number to be sent.
;
; Memory used
;    SendStrIndex,SendStrTmp ( only locally for each call )
; Calls
;    TX_ADD_BUFFER
; Inputs
;    W = String Number
SEND_STRING
    CLRF    SendStrIndex        ;Used as an index into the string table
    MOVWF   SendStrTmp          ;Save a copy of which string we're to write
```

```
SEND_GET_CHAR
    BCF     STATUS,C            ; clear carry (so it dosen't affect byte rotation)
    RLF     SendStrTmp,W        ; Get the saved copy and multiply it by two
    TABLE_JUMP                  ; tablejump macro jumps W number of lines
    ; 'auto' adjust for 256 bytes boundary
    CALL    STRING0             ; Get next character in the string
    GOTO    SEND_CHAR
    CALL    STRING1
    GOTO    SEND_CHAR
    CALL    STRING2
    GOTO    SEND_CHAR
    CALL    STRING3
    GOTO    SEND_CHAR
    CALL    STRING4
SEND_CHAR
    XORLW   0xFF                ; Re-entry point for computed goto's.
    BTFSC   STATUS,Z            ; If the returned byte is 0xFF, end reached
     RETURN
    CALL    TX_ADD_BUFFER       ; Send a single character
    INCF    SendStrIndex,F      ; Point to the next character in the string
    GOTO    SEND_GET_CHAR       ; get the next one

STRING0
    MOVF    SendStrIndex,W      ; Get the string index
    TABLE_JUMP                  ; tablejump macro jumps W number of lines
                                ; adjust for 256 bytes boundary
;    dt      "HELLO WORLD",0x0D,0x0A,0x00  ;Note the zero termination.
    RETLW   0x01                ;UART transport layer header for command packets
    RETLW   0x03                ; Opcode: 0x0C03
    RETLW   0x0C                ; OGF = 3, OCF = 3
    RETLW   0x00                ;
    RETLW   0xFF                ; Byte to mark termination of this string, not sent
STRING1
    MOVF    SendStrIndex,W
    TABLE_JUMP
    RETLW   0x01;               ;UART transport layer header for command packets
    RETLW   0x05;               ;/* Opcode: 0x1005 */
    RETLW   0x10;               ;/* OGF = 4, OCF = 5 */
    RETLW   0x00;               ;/* Param Total Length = 0 */
    RETLW   0xFF                ; Byte to mark termination of this string, not sent
STRING2
    MOVF    SendStrIndex,W
    TABLE_JUMP
    RETLW   0x01;               ;UART transport layer header for command packets
    RETLW   0x33;               ;/* Opcode: 0x0C33 */
    RETLW   0x0C;               ;/* OGF = 3, OCF = 0x33 */
    RETLW   0x07;               ;/* Param Total Length = 7 */
    RETLW   0xFE;               ;/* Host_ACL_Data_Packet_Length */
    RETLW   0x00;               ;/*   (upper byte)             */
    RETLW   0x00;               ;/* Host_SCO_Data_Packet_Length */
    RETLW   0x04;               ;/* Host_Total_Num_ACL_Data_Packets */
    RETLW   0x00;               ;/*   (upper byte)                  */
    RETLW   0x00;               ;/* Host_Total_Num_SCO_Data_Packets */
    RETLW   0x00;               ;/*   (upper byte)                  */
    RETLW   0xFF                ; Byte to mark termination of this string, not sent
STRING3
    MOVF    SendStrIndex,W
    TABLE_JUMP
    RETLW   0x01;               ;UART transport layer header for command packets
    RETLW   0x09;               ;/* Opcode: 0x1009 */
    RETLW   0x10;               ;/* OGF = 4, OCF = 9 */
    RETLW   0x00;               ;/* Param Total Length = 0 */
    RETLW   0xFF                ; Byte to mark termination of this string, not sent
STRING4
    MOVF    SendStrIndex,W
    TABLE_JUMP
    RETLW   0x01;               ; UART transport layer header for command packets
    RETLW   0x05;               ; Opcode: 0x0405
    RETLW   0x04;               ; OGF = 1, OCF = 5
    RETLW   0x0D;               ; Param Total Length = 13
    RETLW   0x55;               ; BD_ADDR_Remote[5] (LSB)
    RETLW   0x55;               ; BD_ADDR_Remote[4]
    RETLW   0x55;               ; BD_ADDR_Remote[3]
    RETLW   0x55;               ; BD_ADDR_Remote[2]
    RETLW   0x55;               ; BD_ADDR_Remote[1]
    RETLW   0x55;               ; BD_ADDR_Remote[0] (MSB)
    RETLW   0x08;               ; Packet_Type LSB
    RETLW   0x00;               ; Packet_Type MSB
    RETLW   0x00;               ; Page_Scan_Rep_Mode
    RETLW   0x00;               ; Page_Scan_Mode
    RETLW   0x00;               ; Clock_Offset LSB
    RETLW   0x00;               ; Clock_Offset MSB
    RETLW   0x00;               ; (Dis)Allow_Role_Switch
    RETLW   0xFF                ; Byte to mark termination of this string, not sent

  END
```

Appendix D Computer Software Listing

```
// BlueTooth.h: interface for the CBlueTooth class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_BLUETOOTH_H__1BF9B960_2515_11D6_9F19_0020CB10A7ED__INCLUDED_)
#define AFX_BLUETOOTH_H__1BF9B960_2515_11D6_9F19_0020CB10A7ED__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "SerialPort.h"
#include <afxmt.h>

class CBlueTooth : public CObject
{
public:

	bool ReadFromModule(int& HCI_Command, int& Status, CSerialPort &m_SerialPort);
	bool WriteToModule(int HCI_Command, CSerialPort& m_SerialPort);
	UCHAR m_Remote_BD_Addr[6];
	UCHAR m_Local_BD_Addr[6];
	UCHAR InputDataBuffer[18];
	UCHAR OutPutDataBuffer[18];
	UCHAR m_HC_ACL_Data_Packet_Length[2];
	UCHAR m_Total_Num_ACL_Data_Packets[2];
	CBlueTooth();
	virtual ~CBlueTooth();

protected:
	UCHAR m_ConnectionHandle[2];

	int m_HCI_Command_Queue_Length;
	UCHAR m_Command_Buffer[32];
	UCHAR m_Event_Buffer[64];
};

// HCI command list
enum HCI_COMMANDS
{
	HCI_RESET,
	HCI_READ_BD_ADDR,
	HCI_READ_BUFFER_SIZE,
	HCI_SET_EVENT_FILTER,
	HCI_PAGE_SCAN_ENABLE,
	HCI_PAGE_SCAN_DISABLE,
	HCI_CREATE_CONNECTION,
	HCI_DISCONNECT,
	HCI_WRITE_DATA,
	HCI_CONNECTION_COMPLETE_EVENT,
	HCI_CONNECTION_COMMAND_STATUS,
	HCI_DISCONNECTION_COMPLETE_EVENT,
	HCI_DATA_PACKET
};

// HCI events we expect
const UCHAR HCI_CONNECTION_COMPLETE = 0x03;
const UCHAR HCI_DISCONNECTION_COMPLETE = 0x05;
const UCHAR HCI_COMMAND_COMPLETE = 0x0e;
const UCHAR HCI_COMMAND_STATUS = 0x0f;



#endif // !defined(AFX_BLUETOOTH_H__1BF9B960_2515_11D6_9F19_0020CB10A7ED__INCLUDED_)
```

```
// BlueTooth.cpp: implementation of the CBlueTooth class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "BluetoothRobot.h"
#include "BlueTooth.h"


#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////


CBlueTooth::CBlueTooth()
{
	m_HCI_Command_Queue_Length = 1; // Start the Queue length at 1 and change if it is larger
	for(int i = 0; i < 6; i++) m_Local_BD_Addr[i] = 0x00; //initialize the local address to 0
	for(i = 0; i < 6; i++) m_Remote_BD_Addr[i] = 0x00; //initialize the Remote address to 0
}

CBlueTooth::~CBlueTooth()
{

}

bool CBlueTooth::WriteToModule(int HCI_Command, CSerialPort& m_SerialPort)
{
	bool ReturnResult = false;
	if(m_HCI_Command_Queue_Length < 1) return(ReturnResult);// The module must be able to accept a
command

	switch(HCI_Command)
	{

		case HCI_RESET:
			//HCI_Reset Command
			m_Command_Buffer[0] = 4; //Bytes to go!
			m_Command_Buffer[1] = 0x01; // command packet
			m_Command_Buffer[2] = 0x03; // op code lsB
			m_Command_Buffer[3] = 0x0c; // op code msB
			m_Command_Buffer[4] = 0x00; // op code command parameters(bytes)
			break;

	   case HCI_READ_BD_ADDR:
			// HCI_Read_BD_ADDR read the local BT address!
			m_Command_Buffer[0] = 4; //Bytes to go!
			m_Command_Buffer[1] = 0x01; // command packet
			m_Command_Buffer[2] = 0x09; // op code lsB
			m_Command_Buffer[3] = 0x10; // op code msB
			m_Command_Buffer[4] = 0x00; // op code command parameters(bytes)
			break;

		   case HCI_READ_BUFFER_SIZE:
		   // HCI_Read_BD_ADDR read the local BT address!
			m_Command_Buffer[0] = 4; //Bytes to go!
			m_Command_Buffer[1] = 0x01; // command packet
			m_Command_Buffer[2] = 0x05; // op code lsB
			m_Command_Buffer[3] = 0x10; // op code msB
			m_Command_Buffer[4] = 0x00; // op code command parameters(bytes)
			break;

		case HCI_SET_EVENT_FILTER:
			//		HCI_SET_EVENT_FILTER;
			m_Command_Buffer[0] = 13; //Bytes to go!
			m_Command_Buffer[1] = 0x01; // command packet
			m_Command_Buffer[2] = 0x05; // op code lsB
			m_Command_Buffer[3] = 0x0c; // op code msB
			m_Command_Buffer[4] = 0x09; // number of parameter bytes to follow
```

```
        m_Command_Buffer[5] = 0x02; // Filter Type connection setup
        m_Command_Buffer[6] = 0x02; // Filter condition type allow a specific BD ADDR
        m_Command_Buffer[7] = m_Remote_BD_Addr[5]; // BD ADDR lsb
        m_Command_Buffer[8] = m_Remote_BD_Addr[4];
        m_Command_Buffer[9] = m_Remote_BD_Addr[3];
        m_Command_Buffer[10] = m_Remote_BD_Addr[2];
        m_Command_Buffer[11] = m_Remote_BD_Addr[1];
        m_Command_Buffer[12] = m_Remote_BD_Addr[0];  //BD ADDR msb
        m_Command_Buffer[13] =0x02;//Auto accept flag autoaccept with role switch disabled
        break;

case HCI_PAGE_SCAN_ENABLE:
        //Now Set to Page Scan Mode
        //HCI_Write_Scan_Enable command
        m_Command_Buffer[0] = 5; //Bytes to go!
        m_Command_Buffer[1] = 0x01; // command packet
        m_Command_Buffer[2] = 0x1a; // op code lsB
        m_Command_Buffer[3] = 0x0c; // op code msB
        m_Command_Buffer[4] = 0x01; // op code command parameters(bytes)
        m_Command_Buffer[5]=0x02;//Scan_Enable->Enable page scan, disable inquiry scan
        break;

case HCI_PAGE_SCAN_DISABLE:
        //Now Disable Page Scan Mode
        //HCI_Write_Scan_Enable command
        m_Command_Buffer[0] = 5; //Bytes to go!
        m_Command_Buffer[1] = 0x01; // command packet
        m_Command_Buffer[2] = 0x1a; // op code lsB
        m_Command_Buffer[3] = 0x0c; // op code msB
        m_Command_Buffer[4] = 0x01; // op code command parameters(bytes)
        m_Command_Buffer[5] = 0x00; // Scan_Enable -> Disable page and inquiry scan
        break;

case HCI_CREATE_CONNECTION:
        //Now Set to Page Scan Mode
        m_Command_Buffer[0] = 17; //Bytes to go!
        m_Command_Buffer[1] = 0x01; // command packet
        m_Command_Buffer[2] = 0x05; // op code lsB
        m_Command_Buffer[3] = 0x04; // op code msB
        m_Command_Buffer[4] = 0x0d; // op code command parameters(bytes)
        m_Command_Buffer[5] = m_Remote_BD_Addr[5]; //remote BT device lsb
        m_Command_Buffer[6] = m_Remote_BD_Addr[4];
        m_Command_Buffer[7] = m_Remote_BD_Addr[3];
        m_Command_Buffer[8] = m_Remote_BD_Addr[2];
        m_Command_Buffer[9] = m_Remote_BD_Addr[1];
        m_Command_Buffer[10] = m_Remote_BD_Addr[0]; // remote BT device msb
        m_Command_Buffer[11] = 0x08; //Packet Type lsb
        m_Command_Buffer[12] = 0x00; //Packet Type msb -> 0x0008 mode DM1
        m_Command_Buffer[13] = 0x00; //Page Scan Repetition mode -> 0x00 mode R0
        m_Command_Buffer[14] = 0x00; //Page Scan Mode -> 0x00 Mandatory Page Scan
        m_Command_Buffer[15] = 0x00; //Clock offset lsb:This is an offset from calling to
                                       //receiving  BT module.
        m_Command_Buffer[16] = 0x00; //Clock offset msb: It would be estimated from the
                                      // results of an Inquiry scan, but will work with 0.
        m_Command_Buffer[17] = 0x00; //Allow role switch -> 0x00 Local device will be
                                       // master and not allow role switch
        break;

case HCI_DISCONNECT:
        //Now Set to Page Scan Mode
        m_Command_Buffer[0] = 7; //Bytes to go!
        m_Command_Buffer[1] = 0x01; // command packet
        m_Command_Buffer[2] = 0x1a; // op code lsB
        m_Command_Buffer[3] = 0x0c; // op code msB
        m_Command_Buffer[4] = 0x03; // op code command parameters(bytes)
        m_Command_Buffer[5] = m_ConnectionHandle[1]; // connection handle lsb
        m_Command_Buffer[6] = m_ConnectionHandle[0]; // connection handle msb
        m_Command_Buffer[7] = 0x16;//disconect reason connection terminated by local host.
        break;

case HCI_WRITE_DATA:
        m_Command_Buffer[0] = 23; //Bytes to go!
        m_Command_Buffer[1] = 0x02; // HCI ACL data packet
        m_Command_Buffer[2] = m_ConnectionHandle[1]; // connection handle lsb
```

```
                                m_ConnectionHandle[0] = m_ConnectionHandle[0] & 0x0f;//clear the upper nibble
                                m_ConnectionHandle[0] = m_ConnectionHandle[0] | 0x0f;// set flags , bc  00 , pb  10
                                m_Command_Buffer[3] = m_ConnectionHandle[0]; // connection handle msb
                                m_Command_Buffer[4] = 0x11; // Data length lsb
                                m_Command_Buffer[5] = 0x00; // Data length msb
                                m_Command_Buffer[6] = OutPutDataBuffer[0]; // Data start eye element 1
                                m_Command_Buffer[7] = OutPutDataBuffer[1];
                                m_Command_Buffer[8] = OutPutDataBuffer[2];
                                m_Command_Buffer[9] = OutPutDataBuffer[3];
                                m_Command_Buffer[10] = OutPutDataBuffer[4];
                                m_Command_Buffer[11] = OutPutDataBuffer[5];
                                m_Command_Buffer[12] = OutPutDataBuffer[6];
                                m_Command_Buffer[13] = OutPutDataBuffer[7];
                                m_Command_Buffer[14] = OutPutDataBuffer[8];
                                m_Command_Buffer[15] = OutPutDataBuffer[9];
                                m_Command_Buffer[16] = OutPutDataBuffer[10];
                                m_Command_Buffer[17] = OutPutDataBuffer[11];
                                m_Command_Buffer[18] = OutPutDataBuffer[12];
                                m_Command_Buffer[19] = OutPutDataBuffer[13];
                                m_Command_Buffer[20] = OutPutDataBuffer[14];
                                m_Command_Buffer[21] = OutPutDataBuffer[15]; // eye element 16
                                m_Command_Buffer[22] = OutPutDataBuffer[16]; // motor lhs
                                m_Command_Buffer[23] = OutPutDataBuffer[17]; // motor rhs Data end
                                break;

                        default :
                        return(ReturnResult);
                                break;

                }//switch

                CEvent event(FALSE, TRUE);
                OVERLAPPED overlapped;
                ZeroMemory(&overlapped, sizeof(OVERLAPPED));
                overlapped.hEvent = event;

                if (!m_SerialPort.Write(&m_Command_Buffer[1], m_Command_Buffer[0], overlapped))
  {
    DWORD dwBytesWritten;
    WaitForSingleObject(event, INFINITE);
    m_SerialPort.GetOverlappedResult(overlapped, dwBytesWritten, TRUE);
                 if (dwBytesWritten == m_Command_Buffer[0]){
                                ReturnResult = true; // the write works
                                m_HCI_Command_Queue_Length--;
                 }//if
  }

                return(ReturnResult);
}


bool CBlueTooth::ReadFromModule(int& HCI_Command, int& Status, CSerialPort &m_SerialPort)
{
                bool ReturnResult = false;
                CEvent event(FALSE, TRUE);
                OVERLAPPED overlapped;
                ZeroMemory(&overlapped, sizeof(OVERLAPPED));
                overlapped.hEvent = event;
                DWORD dwBytesRead, BytesToRead = 0;
                CString errorText;

                BytesToRead = m_SerialPort.BytesWaiting();

                if (!m_SerialPort.Read(&(BYTE)m_Event_Buffer[0], BytesToRead, overlapped))
  {

    if (WaitForSingleObject(event, 1000) == WAIT_OBJECT_0)
    {
      TRACE(_T("Data was read from the serial port\n"));
      m_SerialPort.GetOverlappedResult(overlapped, dwBytesRead, FALSE);
                        if (dwBytesRead == BytesToRead) ReturnResult = true;
    }
    else
      TRACE(_T("No data was read from the serial port\n"));
  }
```

```
  m_SerialPort.GetOverlappedResult(overlapped, dwBytesRead, FALSE);

        if (dwBytesRead == BytesToRead) ReturnResult = true;

        //Handle the command or data packet
        // is the received a data or HCI event packet
        if ((m_Event_Buffer[0] != 0x04)&&(m_Event_Buffer[0] != 0x02)){
          CString errorText;
          errorText ="The received packet was not a data or event packet, received packet type: /X",
m_Event_Buffer[0];
                AfxMessageBox(errorText);
                return(ReturnResult);
                }//if

        if (m_Event_Buffer[0] == 0x04) // HCI Event
                {
                switch(m_Event_Buffer[1]){
                case HCI_CONNECTION_COMPLETE:
                        if (m_Event_Buffer[2] != 0x0b)
                        {
                        return(ReturnResult); //the number of bytes to be read is not correct for this command
                        }

                        HCI_Command = HCI_CONNECTION_COMPLETE_EVENT; // return the event
                        Status = m_Event_Buffer[3]; // Return the status

                        m_ConnectionHandle[1] = m_Event_Buffer[4]; // Connection handle lsb
                        m_ConnectionHandle[0] = m_Event_Buffer[5]; // Connection handle lsb

                        m_Remote_BD_Addr[5] = m_Event_Buffer[6]; // Remote BD Addr lsb
                        m_Remote_BD_Addr[4] = m_Event_Buffer[7];
                        m_Remote_BD_Addr[3] = m_Event_Buffer[8];
                        m_Remote_BD_Addr[2] = m_Event_Buffer[9];
                        m_Remote_BD_Addr[1] = m_Event_Buffer[10];
                        m_Remote_BD_Addr[0] = m_Event_Buffer[11]; // Remote BD Addr msb

                        if ((m_Event_Buffer[12] == 0x01) && (m_Event_Buffer[13] == 0x00)) //data connection
                                                                                        // no encryption
                        ReturnResult = true;

                        break;
                case HCI_DISCONNECTION_COMPLETE:
                        if (m_Event_Buffer[2] != 0x05)
                        {
                        return(ReturnResult); //the number of bytes to be read is not correct for this command
                        }

                        HCI_Command = HCI_DISCONNECTION_COMPLETE_EVENT; // event handled


                        Status = m_Event_Buffer[6]; // Return the disconnect reason

                        m_ConnectionHandle[0] = 0x00;
                        m_ConnectionHandle[1] = 0x00;

                        if (m_Event_Buffer[3] == 0x00) ReturnResult = true;//the disconnection succeeded

                        break;
                case HCI_COMMAND_COMPLETE:
                        Status = 0x00; //This is the status for a CommandComplete event


                        if ((m_Event_Buffer[5] == 0x10)&&(m_Event_Buffer[4]==0x09)){ // read BD Addr
                                if (m_Event_Buffer[2] != 0x09)
                                {
                                return(ReturnResult); //the number of bytes to be read is not correct
                                }
                                m_HCI_Command_Queue_Length = m_Event_Buffer[3]; //commands able
                                                                        //to be issued
                                HCI_Command = HCI_READ_BD_ADDR;
                                if (m_Event_Buffer[6] == 0x00) ReturnResult = true; // command succeeded

                                m_Local_BD_Addr[5] = m_Event_Buffer[7]; // Local BD Addr lsb
                                m_Local_BD_Addr[4] = m_Event_Buffer[8];
                                m_Local_BD_Addr[3] = m_Event_Buffer[9];
```

```
            m_Local_BD_Addr[2] = m_Event_Buffer[10];
            m_Local_BD_Addr[1] = m_Event_Buffer[11];
            m_Local_BD_Addr[0] = m_Event_Buffer[12]; // Local BD Addr msb

            //Set the remote BD Addr to the other board!
            if (m_Event_Buffer[7] == 0x46) m_Remote_BD_Addr[5] = 0x7e;// Remote
                                                                    // Addr lsb
            else m_Remote_BD_Addr[5] = 0x46;
            m_Remote_BD_Addr[4] = m_Event_Buffer[8];
            m_Remote_BD_Addr[3] = m_Event_Buffer[9];
            m_Remote_BD_Addr[2] = m_Event_Buffer[10];
            m_Remote_BD_Addr[1] = m_Event_Buffer[11];
            m_Remote_BD_Addr[0] = m_Event_Buffer[12]; // Remote  Addr msb

        }//if


        else if ((m_Event_Buffer[5] == 0x10)&&(m_Event_Buffer[4]==0x05)){ // read buffer size
            if (m_Event_Buffer[2] != 0x0b)
            {
            return(ReturnResult); //the number of bytes to be read is not correct
                    }
            m_HCI_Command_Queue_Length = m_Event_Buffer[3]; //commands
                                                            //able to be issued
            HCI_Command = HCI_READ_BUFFER_SIZE;
            if (m_Event_Buffer[6] == 0x00) ReturnResult = true; // success

            m_HC_ACL_Data_Packet_Length[0]=m_Event_Buffer[8]; //Max Length ACL
                                                            // data packets
            m_HC_ACL_Data_Packet_Length[1] = m_Event_Buffer[7]; // the module
                                                            // can accept
            m_Total_Num_ACL_Data_Packets[0] = m_Event_Buffer[11]; //  number of
                                                            // data packets
            m_Total_Num_ACL_Data_Packets[1] = m_Event_Buffer[10]; // the module
                                                            // can store
        }//if


        else if ((m_Event_Buffer[5] == 0x0c)&&(m_Event_Buffer[4]==0x05)){ // set event filter
            if (m_Event_Buffer[2] != 0x04)
            {
            return(ReturnResult); //the number of bytes to be read is not correct
            }
            m_HCI_Command_Queue_Length = m_Event_Buffer[3]; //commands
                                                            //able to be issued

            HCI_Command = HCI_SET_EVENT_FILTER;
            if (m_Event_Buffer[6] == 0x00) ReturnResult = true; // success

        }//else if


        else if ((m_Event_Buffer[5] == 0x0c)&&(m_Event_Buffer[4]==0x1a)){ // page scan
            if (m_Event_Buffer[2] != 0x04)                              //enable
            {
            return(ReturnResult); //the number of bytes to be read is not correct
                    }
            m_HCI_Command_Queue_Length = m_Event_Buffer[3]; // commands
                                                            //able to be issued
            HCI_Command = HCI_PAGE_SCAN_ENABLE;
            if (m_Event_Buffer[6] == 0x00) ReturnResult = true; // success

        }//else if

        else if ((m_Event_Buffer[5] == 0x0c)&&(m_Event_Buffer[4]==0x03)){ // HCI Reset
            if (m_Event_Buffer[2] != 0x04)
            {
            return(ReturnResult); //the number of bytes to be read is not correct
                    }
            m_HCI_Command_Queue_Length = m_Event_Buffer[3]; //commands
                                                            // able to be issued
            HCI_Command = HCI_RESET;
            if (m_Event_Buffer[6] == 0x00) ReturnResult = true; // succes

        }//else if
```

```
                    else{
                            errorText ="The unhandled received command was: /X /X",
                                    m_Event_Buffer[5], m_Event_Buffer[4];
                            AfxMessageBox(errorText);
                            return(ReturnResult);
                    }//else
                    break;


            case HCI_COMMAND_STATUS:
                    Status = 0x01; //This is the status for a CommandStatus event
                    m_HCI_Command_Queue_Length = m_Event_Buffer[4]; //commands able to be
                                                                    // issued

                            if ((m_Event_Buffer[6] == 0x04)&&(m_Event_Buffer[6]==0x05)){ //page scan
                                                                                    // enable
                            if (m_Event_Buffer[2] != 0x04)

                            {
                            return(ReturnResult); //the number of bytes to be read is not correct
                            }

                            HCI_Command = HCI_CONNECTION_COMMAND_STATUS;
                            if (m_Event_Buffer[3] == 0x00) ReturnResult = true; // success

                    }//else if


                    else{
                            errorText ="The unhandled received command was: /X /X",
                                    m_Event_Buffer[5], m_Event_Buffer[4];
                            AfxMessageBox(errorText);
                            return(ReturnResult);
                    }//else
                    break;


            default :

                    errorText ="The unhandled received event was: /X", m_Event_Buffer[1];
                    AfxMessageBox(errorText);
                    return(ReturnResult);
                    break;
            }//switch
}//if

else //HCI ACL Data packet
{
HCI_Command = HCI_DATA_PACKET;


            InputDataBuffer[0] = m_Event_Buffer[6]; // Data start eye element 1
            InputDataBuffer[1] = m_Event_Buffer[7];
            InputDataBuffer[2] = m_Event_Buffer[8];
            InputDataBuffer[3] = m_Event_Buffer[9];
            InputDataBuffer[4] = m_Event_Buffer[10];
            InputDataBuffer[5] = m_Event_Buffer[11];
            InputDataBuffer[6] = m_Event_Buffer[12];
            InputDataBuffer[7] = m_Event_Buffer[13];
            InputDataBuffer[8] = m_Event_Buffer[14];
            InputDataBuffer[9] = m_Event_Buffer[15];
            InputDataBuffer[10] = m_Event_Buffer[16];
            InputDataBuffer[11] = m_Event_Buffer[17];
            InputDataBuffer[12] = m_Event_Buffer[18];
            InputDataBuffer[13] = m_Event_Buffer[19];
            InputDataBuffer[14] = m_Event_Buffer[20];
            InputDataBuffer[15] = m_Event_Buffer[21]; // eye element 16
            InputDataBuffer[16] = m_Event_Buffer[22]; // motor lhs
            InputDataBuffer[17] = m_Event_Buffer[23]; // motor rhs Data end

}//else
```

```
        return(ReturnResult);
}
```

```
// BluetoothRobotDlg.h : header file
//

#if !defined(AFX_BLUETOOTHROBOTDLG_H__0C2228A6_1EFE_11D6_9F19_0020CB10A7ED__INCLUDED_)
#define AFX_BLUETOOTHROBOTDLG_H__0C2228A6_1EFE_11D6_9F19_0020CB10A7ED__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "serialport.h"
#include "BlueTooth.h"
#include <fstream.h>

// Eye definitions
#define EYEPIXELWIDTH 24
#define EYEPIXELHEIGHT 24
#define BMPWIDTH 24
#define BMPHEIGHT 24
#define HORIZONTALPIXELS 4
#define VERTICALPIXELS 4

/////////////////////////////////////////////////////////////////////////////
// CBluetoothRobotDlg dialog

class CBluetoothRobotDlg : public CDialog
{
// Construction
public:
        CBitmap m_bmp;
        ofstream m_DataLog;
        void ReadFromBluetooth(BYTE *);
        CSerialPort m_SerialPort;
        UCHAR m_HCI_Data [100];
        UCHAR m_HCI_Reset [20];
        DWORD m_dwMask;
        CBlueTooth TheModule;
        CBluetoothRobotDlg(CWnd* pParent = NULL);       // standard constructor

// Dialog Data
        //{{AFX_DATA(CBluetoothRobotDlg)
        enum { IDD = IDD_BLUETOOTHROBOT_DIALOG };
        CString    m_InputData;
        CString    m_OutputC;
        CString    m_LeftVoltage;
        CString    m_RightVoltage;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CBluetoothRobotDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);          // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;


        struct offset {
                public:
    int xoffset;
    int yoffset;
  };

        offset offsetvalues[256];
        int m_eyevalues[HORIZONTALPIXELS][VERTICALPIXELS];


        // Generated message map functions
        //{{AFX_MSG(CBluetoothRobotDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnChangeInput();
```

```
		virtual void OnOK();
		virtual void OnCancel();
		afx_msg void OnConnect();
		afx_msg void OnTimer(UINT nIDEvent);
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
private:
		void Pause(clock_t waitTime);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_BLUETOOTHROBOTDLG_H__0C2228A6_1EFE_11D6_9F19_0020CB10A7ED__INCLUDED_)
```

```
// BluetoothRobotDlg.cpp : implementation file
//

#include "stdafx.h"
#include "BluetoothRobot.h"
#include <afxmt.h>
#include "BluetoothRobotDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBluetoothRobotDlg dialog

CBluetoothRobotDlg::CBluetoothRobotDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CBluetoothRobotDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CBluetoothRobotDlg)
	m_InputData = _T("");
	m_OutputC = _T("");
	m_LeftVoltage = _T("");
	m_RightVoltage = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

```
void CBluetoothRobotDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBluetoothRobotDlg)
	DDX_Text(pDX, IDC_INPUT, m_InputData);
	DDX_Text(pDX, IDC_OUTPUT, m_OutputC);
	DDX_Text(pDX, IDC_LEFT_VOLTAGE, m_LeftVoltage);
	DDX_Text(pDX, IDC_RIGHT_DIRECTION, m_RightVoltage);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CBluetoothRobotDlg, CDialog)
	//{{AFX_MSG_MAP(CBluetoothRobotDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_EN_CHANGE(IDC_INPUT, OnChangeInput)
	ON_BN_CLICKED(IDC_CONNECT, OnConnect)
	ON_WM_TIMER()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBluetoothRobotDlg message handlers

BOOL CBluetoothRobotDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	time_t ltime;
	time( &ltime );

	m_DataLog.open("c:\\EyeData.txt", ofstream::out);
	m_DataLog << "Bluetooth data trial" << endl;
	m_DataLog <<"The date and time is: " << ctime(&ltime) << endl << endl;

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

 try
 {
	m_SerialPort.Open(1, 57600,  CSerialPort::NoParity, 8, CSerialPort::OneStopBit,
	                             CSerialPort::CtsRtsFlowControl, TRUE);
	m_SerialPort.Setup(128, 128); //Set Port buffer
	m_SerialPort.GetMask(m_dwMask);
 }
 catch (CSerialException* pEx)
 {
   TRACE(_T("Handle Exception, Message:%s\n"), pEx->GetErrorMessage());
   pEx->Delete();
 }

//Initialize the eye data to 0

	for(int i = 0; i < HORIZONTALPIXELS; i++){
		for(int j = 0; j < VERTICALPIXELS; j++){
			m_eyevalues[i][j] = 0;
		}
```

```
		}

		// the squares can be any size smaller than the largest block size set max width and height
		for(i = 0; i < 256; i++){
			offsetvalues[i].xoffset = ((i % 16) * BMPWIDTH);
			offsetvalues[i].yoffset = ((i / 16) * BMPHEIGHT);
		}
		m_bmp.LoadBitmap(IDB_BITMAP1);

		return TRUE;  // return TRUE  unless you set the focus to a control
}

void CBluetoothRobotDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
		if ((nID & 0xFFF0) == IDM_ABOUTBOX)
		{
			CAboutDlg dlgAbout;
			dlgAbout.DoModal();
		}
		else
		{
			CDialog::OnSysCommand(nID, lParam);
		}
}


void CBluetoothRobotDlg::OnPaint()
{
		if (IsIconic())
		{
			CPaintDC dc(this); // device context for painting

			SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

			// Center icon in client rectangle
			int cxIcon = GetSystemMetrics(SM_CXICON);
			int cyIcon = GetSystemMetrics(SM_CYICON);
			CRect rect;
			GetClientRect(&rect);
			int x = (rect.Width() - cxIcon + 1) / 2;
			int y = (rect.Height() - cyIcon + 1) / 2;

			// Draw the icon
			dc.DrawIcon(x, y, m_hIcon);
		}
		else
		{

		CPaintDC dc(this);
		CBitmap *poldbmp;
		CDC memdc;
		CRect ClientRect;

		GetClientRect(&ClientRect);

		int leftside, topside;
		leftside = ClientRect.left + 190;
		topside = ClientRect.top + 101;

		memdc.CreateCompatibleDC(&dc);
		poldbmp = memdc.SelectObject(&m_bmp);

		for(int i = 0; i < HORIZONTALPIXELS; i++){
			for(int j = 0; j < VERTICALPIXELS; j++){
			 dc.BitBlt((leftside +(EYEPIXELWIDTH * i)),
			 (topside+(EYEPIXELHEIGHT*j)),
			 EYEPIXELWIDTH,EYEPIXELHEIGHT, &memdc,
			 offsetvalues[m_eyevalues[i][j]].xoffset,
			 offsetvalues[m_eyevalues[i][j]].yoffset, SRCCOPY);
			}
		}

		memdc.SelectObject(poldbmp);

			CDialog::OnPaint();
```

```
		}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CBluetoothRobotDlg::OnQueryDragIcon()
{
		return (HCURSOR) m_hIcon;
}

void CBluetoothRobotDlg::OnChangeInput()
{
		UpdateData(TRUE);
}

void CBluetoothRobotDlg::OnOK()
{

   m_SerialPort.ClearWriteBuffer();
   m_SerialPort.ClearReadBuffer();
   m_SerialPort.Flush();
   m_SerialPort.Close();
   m_DataLog.close();
   CDialog::OnOK();
}

void CBluetoothRobotDlg::OnCancel()
{

   m_SerialPort.ClearWriteBuffer();
   m_SerialPort.ClearReadBuffer();
   m_SerialPort.Flush();
   m_SerialPort.Close();
   m_DataLog.close();
  CDialog::OnCancel();
}


void CBluetoothRobotDlg::OnConnect()
{

		BOOL WaitResult;
		bool CommandSuccess;
		DWORD BytesToRead = 0;
		int CommandRead, StatusRead, CommandRetries;
		m_SerialPort.ClearWriteBuffer();


//		reset the HCI module

		while((CommandRetries > 0)  && !CommandSuccess){

		TheModule.WriteToModule(HCI_RESET, m_SerialPort);
		Pause(20); //wait for 20 milliseconds for data to be ready
		BytesToRead = m_SerialPort.BytesWaiting();
		if (BytesToRead == 0){
			WaitResult = m_SerialPort.DataWaiting(INFINITE);
			if(!WaitResult) {
				AfxMessageBox("Waiting timeout for reset!");
				OnCancel();
			}
		}

		if(TheModule.ReadFromModule(CommandRead, StatusRead, m_SerialPort))
			if((CommandRead == HCI_RESET) && (StatusRead == 0)) CommandSuccess = true;

		CommandRetries--;

		}//while

		CommandRetries = 5;
		CommandSuccess = false;

		while((CommandRetries > 0)  && !CommandSuccess){
```

```
		TheModule.WriteToModule(HCI_READ_BD_ADDR, m_SerialPort);
		Pause(20); //wait for 20 milliseconds for data to be ready.
		BytesToRead = m_SerialPort.BytesWaiting();
		if (BytesToRead == 0){
			WaitResult = m_SerialPort.DataWaiting(INFINITE);
			if(!WaitResult) {
				AfxMessageBox("Waiting timeout for Reading BD Address!");
				OnCancel();
			}
		}


		if(TheModule.ReadFromModule(CommandRead, StatusRead, m_SerialPort))
			if((CommandRead == HCI_READ_BD_ADDR) && (StatusRead == 0)) CommandSuccess = true;

		CommandRetries--;
		}//while

// set the window header to the local bdaddr

		char windowHdrStr[59];

		wsprintf(&windowHdrStr[0], "CONNECTED TO: 0x%02X%02X%02X%02X%02X%02X\0",
			TheModule.m_Local_BD_Addr[0],
			TheModule.m_Local_BD_Addr[1],
			TheModule.m_Local_BD_Addr[2],
			TheModule.m_Local_BD_Addr[3],
			TheModule.m_Local_BD_Addr[4],
			TheModule.m_Local_BD_Addr[5]);

		SetWindowText(_T(windowHdrStr));

		CommandRetries = 5;
		CommandSuccess = false;

		while((CommandRetries > 0)  && !CommandSuccess){

		TheModule.WriteToModule(HCI_SET_EVENT_FILTER, m_SerialPort);
		Pause(20); //wait for 20 milliseconds for data to be ready.
		BytesToRead = m_SerialPort.BytesWaiting();
		if (BytesToRead == 0){
			WaitResult = m_SerialPort.DataWaiting(INFINITE);
			if(!WaitResult) {
				AfxMessageBox("Waiting timeout for HCI_SET_EVENT_FILTER!");
				OnCancel();
			}
		}


		if(TheModule.ReadFromModule(CommandRead, StatusRead, m_SerialPort))
			if((CommandRead == HCI_SET_EVENT_FILTER) && (StatusRead == 0)) CommandSuccess =
true;

		CommandRetries--;
		}//while


//Now Set to Page Scan Mode


		CommandRetries = 5;
		CommandSuccess = false;

		while((CommandRetries > 0)  && !CommandSuccess){

		TheModule.WriteToModule(HCI_PAGE_SCAN_ENABLE, m_SerialPort);
		Pause(20); //wait for 20 milliseconds for data to be ready.
		BytesToRead = m_SerialPort.BytesWaiting();
		if (BytesToRead == 0){
			WaitResult = m_SerialPort.DataWaiting(INFINITE);
			if(!WaitResult) {
				AfxMessageBox("Waiting timeout for HCI_PAGE_SCAN_ENABLE!");
				OnCancel();
			}
		}
```

```
		if(TheModule.ReadFromModule(CommandRead, StatusRead, m_SerialPort))
			if((CommandRead == HCI_PAGE_SCAN_ENABLE) && (StatusRead == 0)) CommandSuccess =
true;

		CommandRetries--;
		}//while

		// Enable processing thread for comm
		SetTimer(0,20,NULL);

}


void CBluetoothRobotDlg::OnTimer(UINT nIDEvent)
{
		int BytesToRead;
		int CommandRead, StatusRead;

		BytesToRead = m_SerialPort.BytesWaiting();
			if(BytesToRead > 0){
				if(TheModule.ReadFromModule(CommandRead, StatusRead, m_SerialPort)){
					if(CommandRead == HCI_DISCONNECTION_COMPLETE_EVENT)
					{
						KillTimer(0);
					}

					else if (CommandRead == HCI_DATA_PACKET) {
						//set the values for the eye data
						int k = 0;
						for(int i = 0; i < HORIZONTALPIXELS; i++){
							for(int j = 0; j < VERTICALPIXELS; j++){
								m_eyevalues[i][j] =
									TheModule.OutPutDataBuffer[k];
								k++;
							}
						}
					// now set the voltage for the motors 256 levels max 5 volts
					m_LeftVoltage.Format("%2.2f",
						(TheModule.OutPutDataBuffer[16]/256.0*5.0));
					m_RightVoltage.Format("%2.2f",
						(TheModule.OutPutDataBuffer[17]/256.0*5.0));

					// update the dialog with the new bitmap

					OnPaint();
					}
					else if (CommandRead == HCI_CONNECTION_COMPLETE_EVENT){

					char RemoteAddrStr[59];

					wsprintf(&RemoteAddrStr[0], "0x%02X%02X%02X%02X%02X%02X\0",
					TheModule.m_Remote_BD_Addr[0],
					TheModule.m_Remote_BD_Addr[1],
					TheModule.m_Remote_BD_Addr[2],
					TheModule.m_Remote_BD_Addr[3],
					TheModule.m_Remote_BD_Addr[4],
					TheModule.m_Remote_BD_Addr[5]);

					m_OutputC = RemoteAddrStr;

					BOOL WaitResult;
					bool CommandSuccess;
					DWORD BytesToRead = 0;
					int CommandRead, StatusRead, CommandRetries;


					//Disable Page Scan Mode


					CommandRetries = 5;
					CommandSuccess = false;

					while((CommandRetries > 0)  && !CommandSuccess){
```

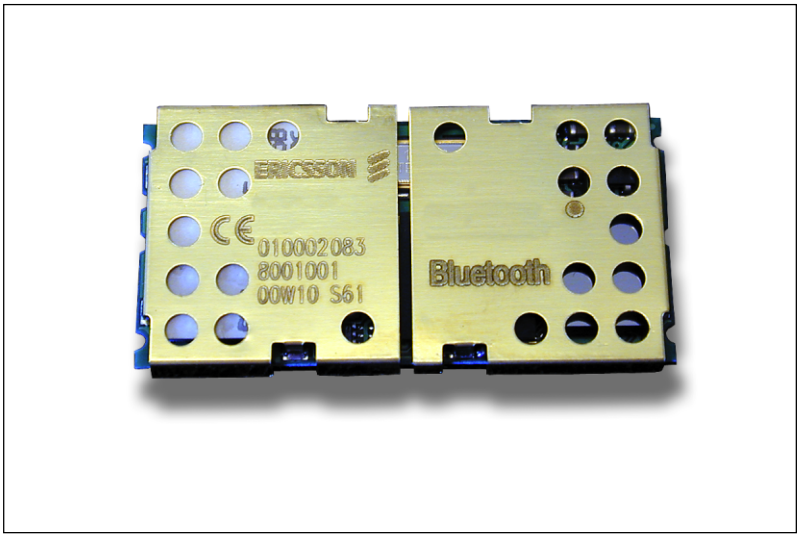
```
                                        TheModule.WriteToModule(HCI_PAGE_SCAN_DISABLE,
                                                m_SerialPort);
                                        Pause(20); //wait for 20 milliseconds for data to be ready.
                                        BytesToRead = m_SerialPort.BytesWaiting();
                                        if (BytesToRead == 0){
                                                WaitResult = m_SerialPort.DataWaiting(INFINITE);
                                                if(!WaitResult) {
                                                        AfxMessageBox("Waiting timeout for
                                                                HCI_PAGE_SCAN_DISABLE!");
                                                        OnCancel();
                                                }
                                        }


                                    if(TheModule.ReadFromModule(CommandRead, StatusRead,
                                                                                    m_SerialPort))
                                        if((CommandRead == HCI_PAGE_SCAN_DISABLE) &&
                                                (StatusRead == 0)) CommandSuccess = true;

                                    CommandRetries--;
                                    }//while


                                    UpdateData(FALSE);
                                    }
                                    else KillTimer(0);
                                }//if
                        }//if


                CDialog::OnTimer(nIDEvent);
}



// implements a pause function for waitTime milliseconds
void CBluetoothRobotDlg::Pause(clock_t waitTime)

{
  clock_t EndTime;
  EndTime = waitTime + clock();
  while(EndTime > clock());
}
```

Appendix E Ericsson ROK101008 Bluetooth Module Datasheet

ROK 101 008
Bluetooth Module

Key Features

- Qualified to Bluetooth 1.0B
- RF output power class 2
- FCC and ETSI approved
- 460 kb/s max data rate over UART
- UART and PCM interface
- I^2C interface
- Internal crystal oscillator
- HCI firmware included
- Point to Point connection
- Built-in shielding



Supported Bluetooth Profiles

- Generic Access Profile
- Service Discovery Application Profile
- Serial Port Profiles
  - Dial-up networking
  - Fax
  - Headset
- Generic Object Exchange Profiles
  - File transfer
  - Object Push
  - Synchronisation

Suggested Applications

- Computers and peripherals
- Handheld devices and accessories
- Access points

Description

ROK 101 008 is a short-range module for implementing Bluetooth functionality into various electronic devices. The module consists of three major parts; a baseband controller, a flash memory, and a radio that operates in the globally available 2.4–2.5 GHz free ISM band.

Both data and voice transmission is supported by the module. Communication between the module and the host controller is carried out via UART and PCM interface.

ROK 101 008, which is compliant with Bluetooth version 1.0B and critical errata, is a Class 2 Bluetooth Module (0 dBm) and is type-approved.

ERICSSON

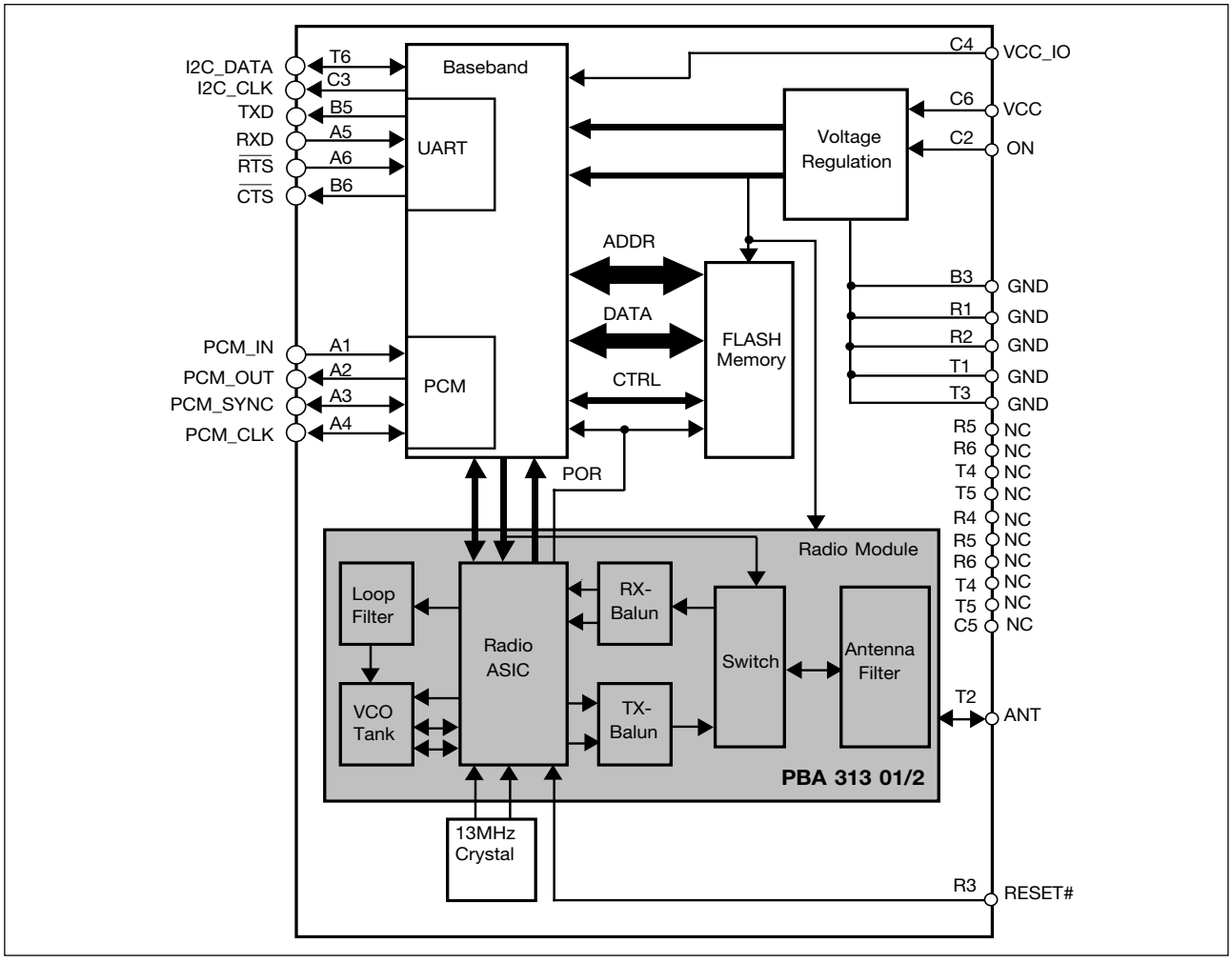


Figure 1. Block Diagram

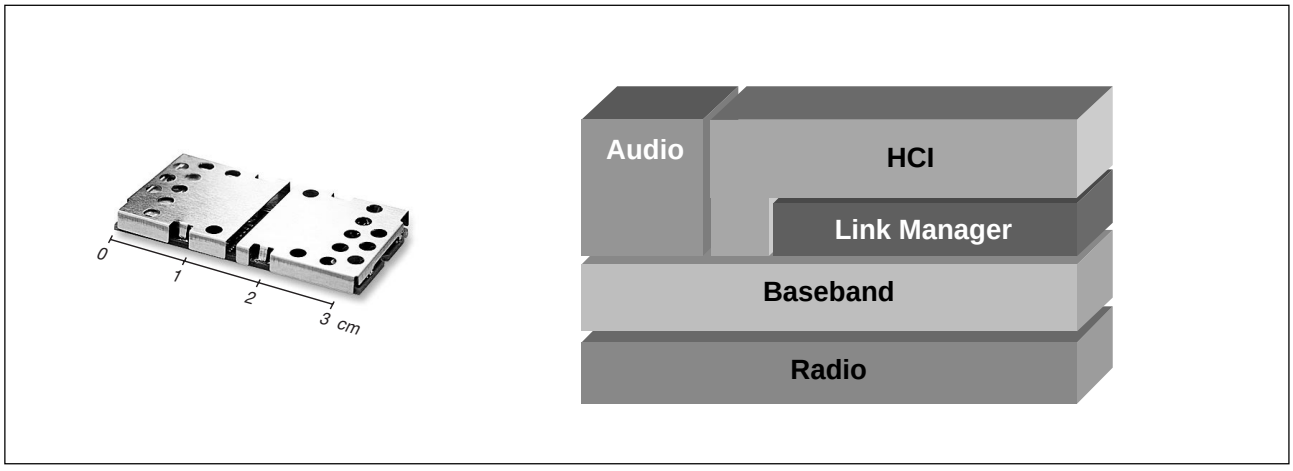


Figure 2. Actual size of the Bluetooth Module and also showing the HW and FW stack.

Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Temperature				
Storage temperature	T_{Stg}	-30	+85	°C
Operating temperature	T_{Amb}	0	+75	°C
Power Supply				
VCC	V_{CC}	-0.3	+5.25	V
VCC_IO	V_{CC_IO}	-0.8	+3.6	V
Digital Inputs				
Input low voltage	V_{IL}	-0.5		V
Input high voltage	V_{IH}		V_{CC_IO} + 0.3	V
Antenna Port				
Input RF power	In-band		15	dBm
	Out of band		15	dBm

Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Unit
Temperature					
Ambient temperature, Test	T_{Amb}		+23		°C
Power Supply					
Positive Supply Voltage	V_{CC}		+3.3		V
I/O Ports Supply Voltage	V_{CC_IO}		+3.3		V

DC Specifications

Unless otherwise noted, the specification applies for Tamb = 0 to +75°C

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Power Supply						
Supply Voltage		V_{CC}	3.175	3.3	5.25	V
I/O Ports Supply Voltage		V_{CC_IO}	2.7	3.3	3.6	V
Digital Inputs						
Logical Input High	Except ON signal	V_{IH1}	0.7xV_{CC_IO}		V_{CC_IO}	V
Logical Input Low	Except ON signal	V_{IL2}	0		0.3xV_{CC_IO}	V
Logical Input High	ON signal only	V_{IH2}	2.0		V_{CC}	V
Logical Input Low	ON signal only	V_{IL2}	0		0.4	V
RESET# Input Low	RESET# signal only	V_{RESET}	0		0.4	V
Digital Outputs						
Logical Output High		V_{OH}	0.9xV_{CC_IO}		V_{CC_IO}	V
Logical Output Low		V_{OL}	0		0.1xV_{CC_IO}	V

Current Consumption

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Average Current Consumption	$I_{CC} + I_{CC\_IO}$					
HW Shutdown state	See note 1	I_{SHW}		1		µA
SW Standby Mode	Can only be woken up via UART see note 3	I_{STA1}		250		µA
	Can be woken up via UART and via RF see notes 2 & 3	I_{STA2a}		1.5		mA
Idle state	After HCI - reset	I_{IDL1}		15		mA
	After a H/W reset	I_{IDL2}		23		mA
Connection Mode	Master Mode	I_{CS_M}		35		mA
	Slave Mode	I_{CS_S}		25		mA
VCC_IO supply		I_{CC_IO}		2		mA

Notes

1. Current consumption is based upon when the pin 'ON ' is low and 'VCC_IO ' is grounded.
2. HCI basic settings have not been sent to UUT.
3. Implemented by using the Ericsson_HCI_Save_Power command. ISTA1 is entered by sending the following command '0123 FC01 03'. ISTA2 is entered by the command '0123 FC01 02'.

Timing Performance

Parameter	Condition	Symbol	Min	Typ	Max	Unit
System start-up time from power on				1250		ms
RESET# signal duration	Sink current > 1mA			1		ms
Firmware timer resolution				6.55		ms

PCM

Parameter	Condition	Symbol	Min	Typ	Max	Unit
PCM clock frequency See fig 3 & 4	Master mode	f_{PCM_CLK}		2000		kHz
	Slave mode	f_{PCM_CLK}	128		2048	kHz
PCM sample rate sync. frequency See fig 3 & 4		f_{PCM_SYNC}		8		kHz
PCM clock high period		t_{CCH}	200			ns
PCM clock low period		t_{CCL}	200			ns
PCM_SYNC (setup) to PCM CLK (fall)	See fig 5	t_{PSS}	100	1000		ns
PCM_SYNC pulse length	See fig 5	t_{PSH}	200	460		ns
PCM_X in (setup) to PCM_CLK (fall)		t_{DSL}	100			ns
PCM_X in (hold) from PCM_CLK (fall)		t_{DSH}	100			ns
PCM_X out valid from PCM_CLK (rise)		t_{PDLP}			150	ns

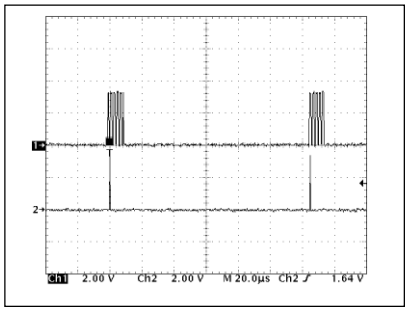


Figure 3.

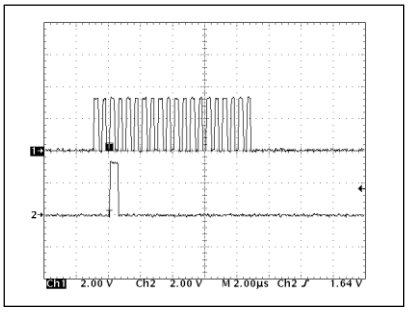


Figure 4.

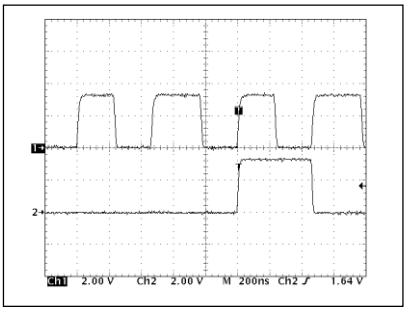


Figure 5.

RF Specifications

General

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Frequency range			2.402		2.480	GHz
Antenna load				50		Ω
VSWR	RX mode			2:1		
VSWR	TX mode, see note 4			2:1		

Notes

4. During the TX mode, the VSWR specification states the limits that are acceptable before any other RF parameters are strongly effected. i.e. frequency deviation and initial frequency error.

Receiver Performance (0.1% BER)

Parameter	**Condition**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Sensitivity level				-75	-70	dBm
Max input level			-20			dBm
Spurious Emissions	30MHz to 1GHz			-74	-57	dBm
Spurious Emissions	1GHz to 12.75GHz			-60	-47	dBm

Transmitter Performance

Parameter	**Condition**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Frequency deviation		f_{MOD}	140	155	170	kHz
TX power			-6	+2	+4	dBm
TX carrier drift	1 slot	$F_{DRIFT(1)}$	-25	+5	+25	kHz
	3 slots	$F_{DRIFT(3)}$	-40	+10	+40	kHz
	5 slots	$F_{DRIFT(5)}$	-40	+15	+40	kHz
20dB bandwidth	Peak detector			750	1000	kHz
Spurious Emissions	30MHz - 1GHz				-36	dBm
	1GHz - 12.75GHz				-30	dBm
	1.8GHz - 1.9GHz				-47	dBm
	5.15GHz - 5.3GHz				-47	dBm

Pin Description				
Pin	**Pin Name**	**Type**	**Direction**	**Description**
A1	PCM_IN	CMOS	In	PCM data, see notes 5,6
A2	PCM_OUT	CMOS	Out	PCM data, see notes 5,6
A3	PCM_SYNC	CMOS	In/Out	Sets the PCM data sampling rate, see notes 5,6
A4	PCM_CLK	CMOS	In/Out	PCM clock that sets the PCM data rate, see notes 5,6
A5	RXD	CMOS	Input	RX data to the UART, see note 6
A6	RTS	CMOS	Input	Flow control signal, Request To Send data from UART, see notes 5,6
B1	NC	-	-	Do not connect
B2	NC	-	-	Do not connect
B3	GND	Power	Power	Signal ground
B4	NC	-	-	Do not connect
B5	TXD	CMOS	Output	TX data from the UART, see note 6
B6	CTS	CMOS	Output	Flow control signal, Clear To Send data from UART, see note 6
C1	NC	-	-	Do not connect
C2	ON	Power	Input	When tied to V_{CC}, the module is enabled.
C3	I²C_CLK	CMOS	Output	I²C clock signal, see note 6
C4	VCC_IO	Power	Power	External supply rail to the Input / Output ports
C5	NC	-	-	Do not connect
C6	VCC	Power	Power	Supply Voltage
R1	GND	Power	Power	Signal ground
R2	GND	Power	Power	Signal ground
R3	RESET#	CMOS	Input	Active low reset, see note 7
R4	NC	-	-	Do not connect
R5	NC	-	-	Do not connect
R6	NC	-	-	Do not connect
T1	GND	Power	Power	Signal Ground
T2	ANT	RF	In/Out	50Ω Antenna connection
T3	GND	Power	Power	Signal Ground
T4	NC	Power	Power	Test point, internal voltage regulator - Do not connect
T5	NC	-	-	Do not connect
T6	I²C_DATA	CMOS	In/Out	I²C data signal, see note 6

Notes

5. 100kΩ pull-up resistors to V_{CC_IO} are incorporated on the module. PCM signals direction is programmable.
6. CMOS buffers are low voltage TTL compatible signals.
7. RESET# signal must be fed from an open drain output.

Mechanical Specification

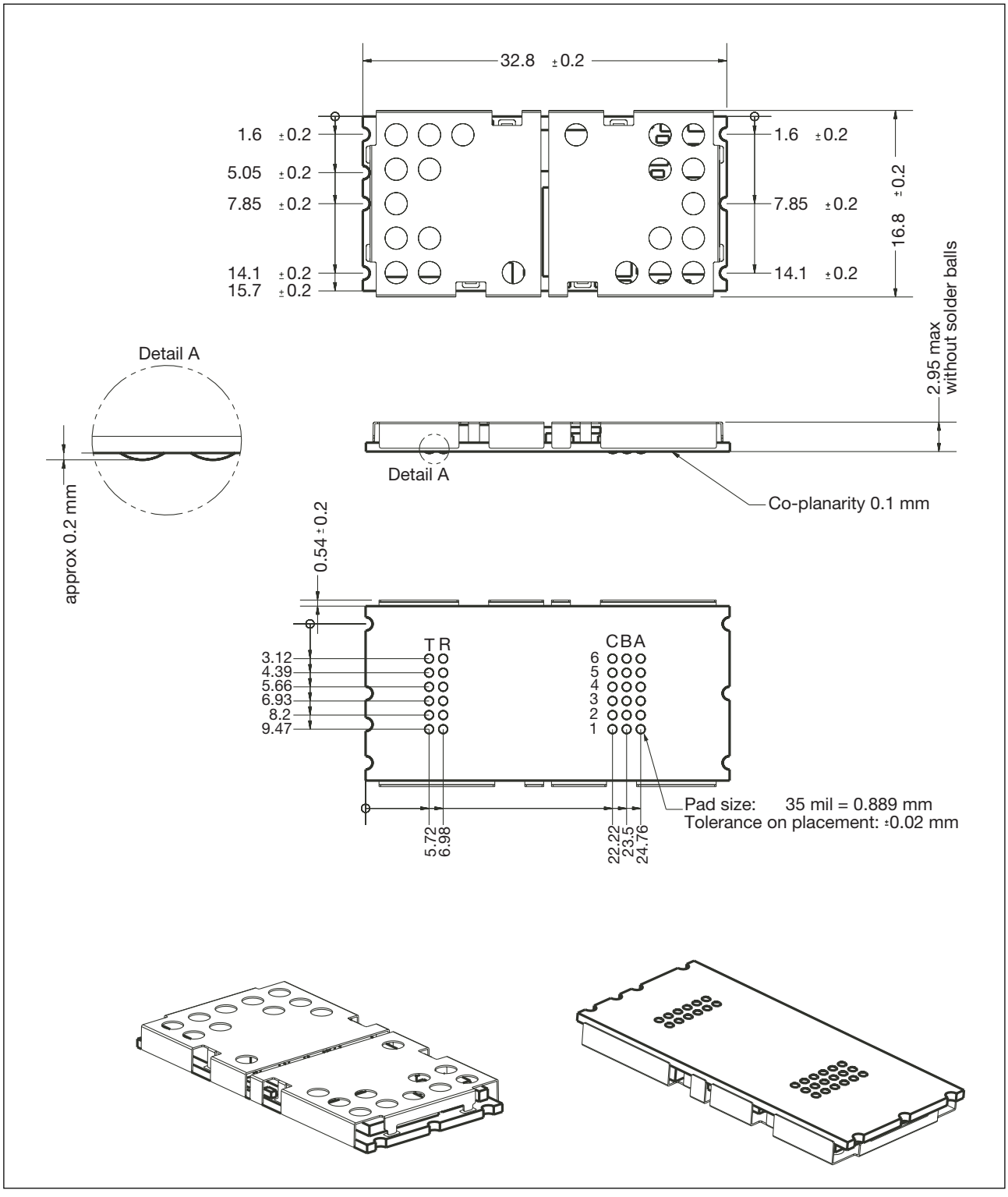


Figure 6. Mechanical dimensions

Functional Description

The ROK 101 008 is a complete Bluetooth module that has been specified and designed according to the Bluetooth System v1.0B . Its implementation is based on a high-performance integrated radio transceiver (PBA 313 01/2) working with a baseband controller, a flash memory and surrounding secondary components.

Block Diagram

ROK 101 008 has five major operational blocks. Figure 7 illustrates the interaction of the various blocks. The functionality of each block is as follows:

1. Radio functionality is achieved by using the Bluetooth Radio, PBA 313 01/ 2. Six operational blocks are shown for the radio section and their operation is as follows:

1a) VCO-tank is a part of the phase locked loop. The modulation is performed directly on the VCO. To ensure high performance the VCO-tank is laser trimmed.

1b) Loop filter, filters the tuning voltage of the VCO-tank.

1c) RX-balun handles transformation from unbalanced to balanced transmission.

1d) TX-balun handles biasing of the output amplifier stage and transformation from balanced to unbalanced transmission.

1e) Antenna switch directs the power either from the antenna filter to the receive ports or from the ASIC output ports to the antenna filter.

1f) Antenna filter band-pass filters the radio signal.

2. The baseband controller is an ARM7-Thumb based chip that controls the operation of the radio transceiver via the UART interface. Additionally, the baseband controller has a PCM Voice and I²C interface. The baseband controller ROP 101 1112/C is used.
3. A Flash memory is used together with the baseband controller. Please, refer also to the Firmware section.
4. The voltage regulation block regulates and filters the supply voltage. V_{CC} is typically 3.3V and two regulated voltages are produced.
5. An internal clock is mounted on the module. The clock frequency is 13MHz and is generated from a crystal oscillator that guarantees a timing accuracy within ± 20ppm.

Bluetooth Module stack

The Host Controller Interface (HCI) handles the communication by the transport layer through the UART interface with the host, see figure 8. The Baseband and radio provides a secure and reliable radio link for higher layers. The following sections describe the Bluetooth module stack in more detail. It is implemented in accordance with and complies with the Specification of the Bluetooth System v1.0B .

Bluetooth Radio Interface

The Bluetooth module is a class 2 device with 4dBm maximum output power with no power control needed. Nominal range of the module with a typical antenna is up to a range of 10 m (at 0 dBm). It is compliant with FCC and ETSI regulations in the ISM band.

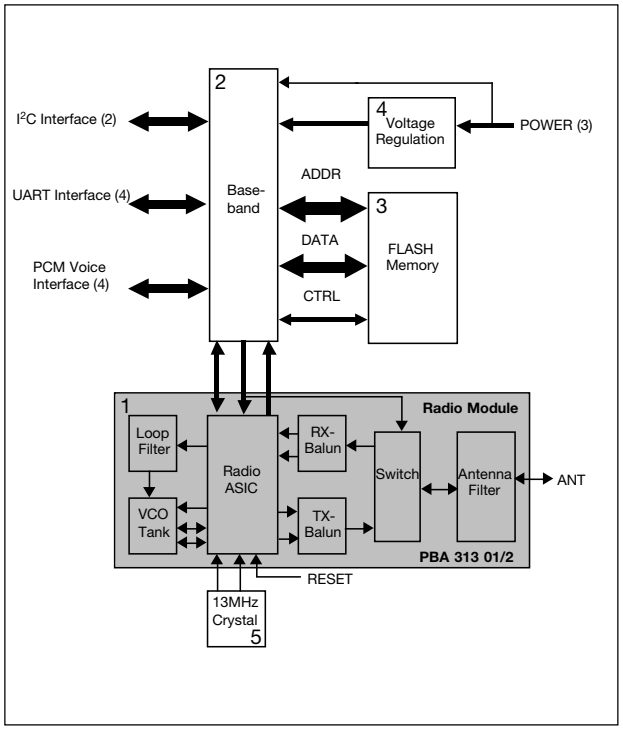


Figure 7. Simplified Block Diagram

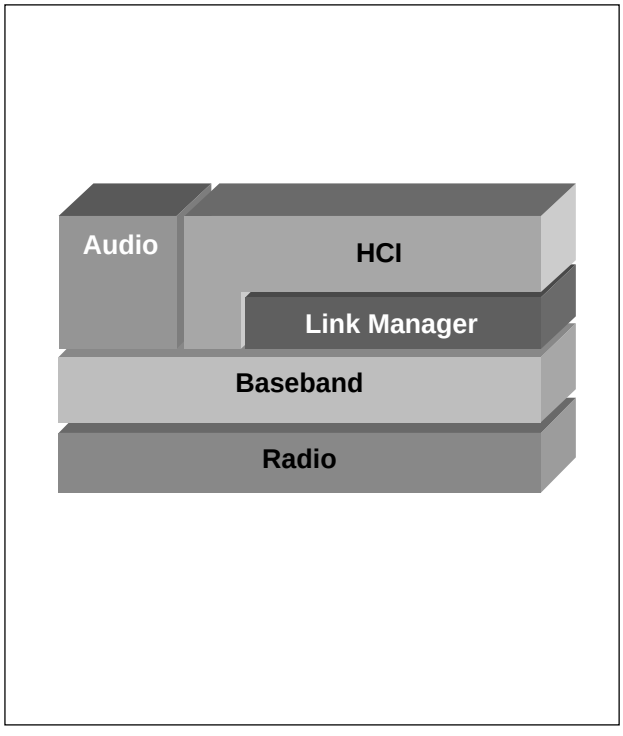


Figure 8. HW/FW parts included in the Ericsson Buetooth module.

Baseband
By default the first unit setting up a connection is the master of the point to point link. The master transmits in the even timeslots and the slave transmits in the odd timeslots.

For full duplex transmission, a Time-Division Duplex (TDD) scheme is used. Packets are sent over the air in timeslots, with a nominal length of 625 µs. A packet can be extended to a maximum of 5 timeslots (DM5 and DH5 packets) and is then sent by using the same RF channel for the entire packet.

Two types of connections are provided - Asynchronous Connectionless Link (ACL) for data and the Synchronous Connection Oriented Link (SCO) for voice. Only two 64kb/s voice channel are supported, HV1 and HV3. Furthermore, there are also packages used for link control purposes.

A variety of different packet types with error correction schemes and data rates can be used over the air interface. Also asymmetric communication is available for high speed communication in one direction.

The Baseband provides the link-setup and control routines for the layers above. Furthermore, the Baseband also provides Bluetooth security like encryption, authentication and key management.

Please refer to the Specification of the Bluetooth System v1.0B part B for in-depth information regarding the Baseband.

Type	User Payload (bytes)	FEC	CRC	Symetric Max. rate	Asymetric Max.rate
ID	na	na	na	na	na
NULL	na	na	na	na	na
POLL	na	na	na	na	na
FHS	18	2/3	yes	na	na

Link control packets

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symetric Max. rate (kb/s)	Asymetric Max rate (kb/s)	
						Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	no	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6

ACL packets

Type	Payload header (bytes)	User Payload (bytes)	FEC	CRC	Symetric Max. rate (kb/s)
HV1	na	10	1/3	no	64.0
HV3	na	30	no	no	64.0
DV	1D	10+(0-9) D	2/3 D	Yes D	64.0+57.6 D

SCO packets

Table 1: Link Control Packets Table, ACL Packets Table, SCO packets

Firmware (FW)

The module includes firmware for the host controller interface, HCI, and the link manager, LM.
The FW resides in the Flash and is available in object code format.

Link Manager (LM)
The Link Manager in each Bluetooth module can communicate with another Link Manager by using the Link Manager Protocol (LMP) which is a peer to peer protocol. The LMP messages have the highest priority and are used for link-setup, security, control and power saving modes. The receiving Link Manager filter-out the message and does not need to acknowledge the message to the transmitting LM due to the reliable link provided by the Baseband and radio.

LM to LM communication can take place without actions taken by the host. Discovery of features at other Bluetooth enabled devices nearby can be found and saved for later use by the host.

Please refer to the Specification of the Bluetooth System v1.0B part C for in-depth information regarding the LMP.

Host Control Interface (HCI)
The HCI provides a uniform command I/F to the Baseband and Link Manager and also to HW status registers.

There are three different types of HCI packets:

- HCI command packets – from host to Bluetooth module HCI.
- HCI event packets – from Bluetooth module HCI to host.
- HCI data packets – going both ways.

It is not necessary to make use of all different commands and events for an application. If the application is aimed

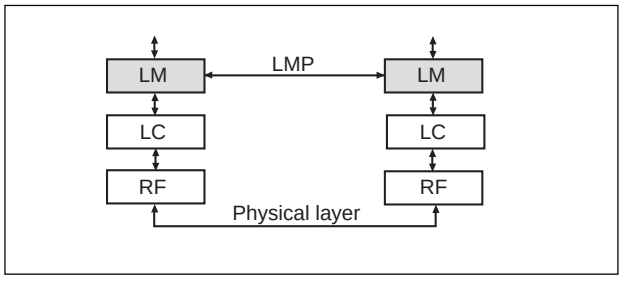


Figure 9. Link manager

at a pre-specified profile, the capabilities of such a profile is necessary to adjust to – see Specification of the Bluetooth System v1.0B Profiles.
The interface for communicating with the Bluetooth module is achieved with the HCI UART Transport Layer on top of HCI, the module will communicate with a host through the UART I/F. The PCM I/F is also available for communicating voice.
Please refer to the Specification of the Bluetooth System v1.0B part H:1-4 for in-depth information regarding the HCI and different transport layers

Module HW Interfaces

UART Interface
The UART implemented on the module is an industry standard 16C450 and supports the following baud rates: 300, 600, 900, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 and 460800 bits/s. 128 byte FIFOs are associated with the UART.

Four signals will be provided for the UART interface. TxD & RxD are used for data flow, and RTS & CTS is used for flow control.

Please refer to the Specification of the Bluetooth System v1.0B part H:4 regarding the HCI and UART transport layers.

PCM Voice Interface
The standard PCM interface has a sample rate of 8 kHz (PCM_SYNC). The PCM clock is variable between 128 kHz and 2.0 MHz in the PCM slave mode. The PCM data can be linear PCM (13-16bit), µ-Law (8bit) or A-Law (8bit).
The PCM I/F can be either master or slave – providing or receiving the PCM_SYNC. Redirection of PCM_OUT and PCM_IN can be accomplished as well.
Over the air the encoding is programmable to be, CVSD and A-Law or µ-Law.

I²C Interface
A master I²C I/F is available on the module. The control of the I²C pins are performed by Ericsson specific HCI commands available in the FW implementation – see Appendix C.

Antenna
The ANT pin should be connected to a 50Ω-antenna interface, thereby supporting the best signal strength performance. Ericsson Microelectronics can recommend application specific antennas – see Appendix C.

RESET#
The assignment of the RESET# input is to generate a reset signal to the complete Bluetooth module. During power-up the reset signal is set 'low' automatically so that power supply glitches are avoided. Therefore no reset input should be required after power-up.

Power-up Sequence

There is no need for a power up sequence if VCC, ON and VCC_IO are tied together.

A power up sequence, if used, shall be applied accordingly: Connection of the supply rails, GND and then V_{CC}; then the ON signal should be applied in order to initiate the internal regulators; and finally, the V_{CC_IO} supply rail can be activated.

The power-down sequence is similar to the power-up procedure but in the reverse format. Therefore, the disconnection of the signals shall be as follows: V_{CC_IO}, ON,V_{CC} and finally GND.

Power
There are three inputs to the Voltage Management section (V_{CC}, V_{CC_IO}, ON). V_{CC} is the supply voltage that is typically 3.3V.

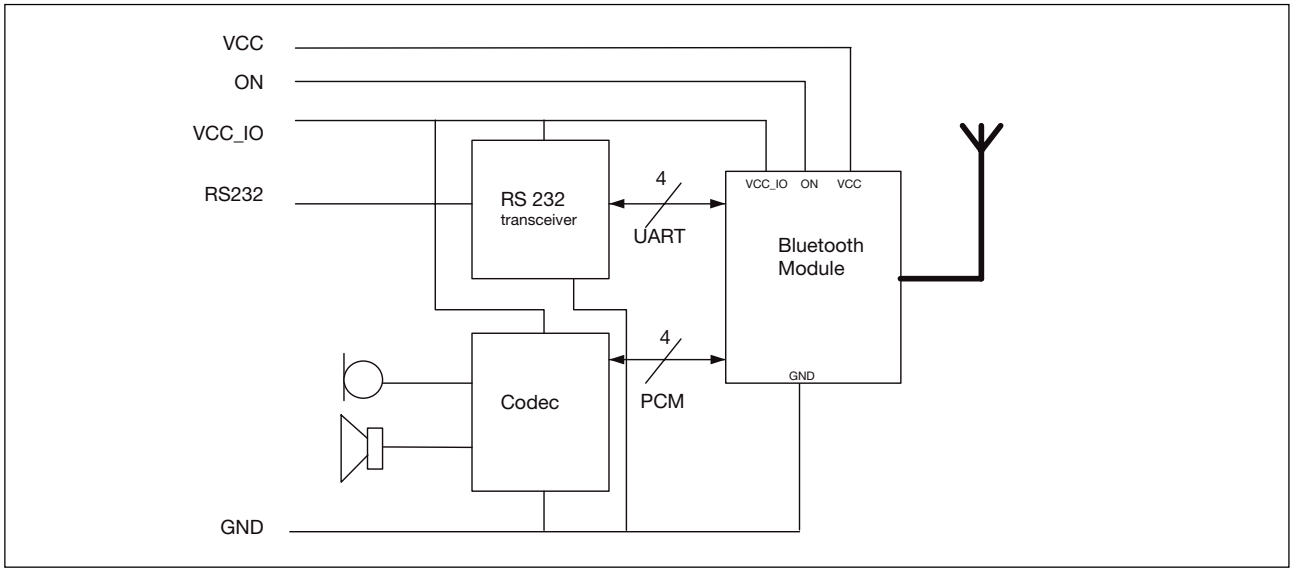


Figure 10. Application block schematics. A typical UART or PCM configuration.

A separate power supply rail (V_{CC_IO}) is provided for the I/O ports, UART and PCM. V_{CC_IO} can either be connected to V_{CC} or to a dedicated supply rail, which is the same as the logical interface of the host.

Shielding / EMC Requirements
The module has its own RF shielding and is approved according to the standards by FCC and ETSI.

If the approval number is not visible on the outside when the module is utilized in the final product, an exterior label must state that there is a transmitter module inside the product.

Ground
Ground should be distributed with very low impedance as a ground plane. Connect all GND pins to the ground plane.

Assembly Guidelines

Solder Paste
The ROK 101 008 module is made for surface mounting and the SSP connection pads have been formed after printing eutectic Tin/Lead solder paste. The solder paste to use is not critical as long as this is a normal eutectic solder paste. A preferred solder paste height is 150µm.

Soldering Profile
It must be noted that the module should not be allowed to be hanging upside down in the re-flow operation. This means that the module has to be assembled on the side of the PCB that is soldered last.

The re-flow process should be a regular surface mount soldering profile (full convection strongly preferred); the ramp-up should not be higher than 2°C/s and with a peak temperature of 210-235°C during 20-60 seconds.

Pad Size
It is recommended that the pads on the PCB should have a diameter of 0.7-0.9 mm. The surface finish on the PCB pads should be Nickel/Gold or a flat Tin/Lead surface or OSP (Organic Surface Protection).

Placement
The placement machine should be able to recognize odd BGA combinations (all ball recognition preferred) and be able to pick the component asymmetrical. The module contains a flat pick-area of 10mm diameter minimum. The weight of the module is typically 2.8gr.

Storage
Keep the component in its dry pack when not yet using the reel. After removal from the dry pack ensure that the modules are soldered onto the PCB within 48 hours.

Marking

Every module is marked with the following information on the:

a) Component designation: “ROK 101 008”.
b) Ericsson’s name and logotype.
c) Manufacturing code (place, year, week) and batch number.
d) CE logotype
e) Type approval RTA no. See manual

Ordering Information

Part No.
ROK 101 008/2

Packaging

All devices will be delivered in a package protecting them from electrostatic discharges and mechanical shock. The package will be marked with the following information:

a) Delivery address.
b) Purchase order-number
c) Type of goods and component designation.
d) Ericsson’s name and logotype.
e) Date of manufacture and batch number.
f) Number of components in the package.

Abbreviations

ASIC - Application Specific Integrated Circuit
BER - Bit Error Rate
CMOS - Complementary Metal Oxide Semiconductor
C/I - Carrier to Interference Ratio
DCE - Data Circuit terminating Equipment
GP - Gold Print
HCI - Host Controller Interface
ISM - Industrial Scientific and Medical
PCB - Printed Circuit Board
PCM - Pulse Code Modulation
PDA - Personal Digital Assistant
PtP - Point to Point
Rx - Receive
SIG - Special Interest Group
SSP - Screen Solder Print
Tx - Transmit
UART - Universal Asynchronous Receiver Transmitter
VCO - Voltage Controlled Oscillator

APPENDIX A

Getting Started

The ROK 101 008 Bluetooth module is easy to use when designing a Bluetooth application. However, there is a need for know-how in the Bluetooth System specification v1.0B as well as the Profile specification v1.0B when designing and end-customer product. The list below show some parts that would make designing convenient.

- Bluetooth module
- Know-how in Bluetooth specification regarding HCI commands
- Test board with UART/PCM or USB I/F
- Visual C++ for PC SW design
- Preferably HCIdriver, L2CAP, RFCOMM and SDP from Ericsson

All information needed, regarding how to drive the HCI over UART is specified in part H4 of the Bluetooth System v1.1 further more part H1 and also Appendix IX with message charts is relevant.

Below follow an example of how to set up an ACL link between to Bluetooth modules by using the UART I/F and also a schematic of how to interface the module and control it by a host, normally a PC or microcontroller.

Principle schematic for UART inter-connect

The inter-connection to the level-shifter when designing a test-board could be according to the schematic below. (Figure A1.)

The Bluetooth module can be connected as a DCE/DTE and a modem/nullmodem cable could therefor be used in-between the test-board and the PC.

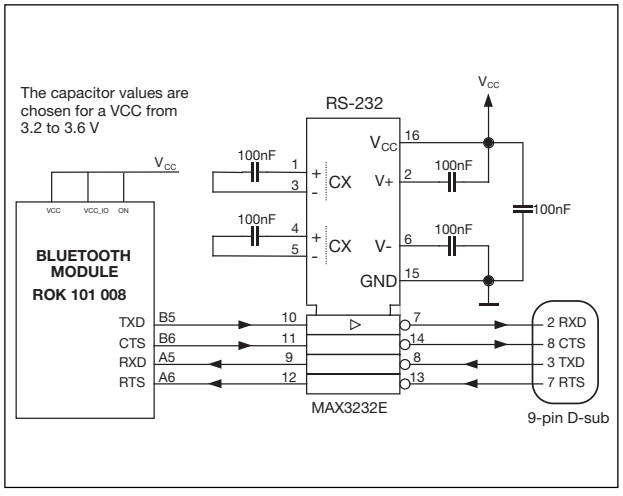


Figure A1. Bluetooth module connected as a DCE through level shifter

Setting up a Bluetooth point-to-point connection

The Host Controller Interface (HCI) in the module is a command I/F. The host presents commands to the HCI and receives events back from the HCI of the module. The module Link Manager provides link set up capability to the HCI.

- Host_B Bluetooth module is set in paging scan mode – listening for a Bluetooth device asking for a new connection
- Host_A Bluetooth module is set in paging mode asking for a connection to Host_B

This is accomplished by first setting up the connection between the Host and the module and thereafter creating the connection between the modules using HCI commands.

Host set-up via UART:

There are 4 different types of HCI-packets accepted on the UART I/F.

HCI packet type	HCI packet indicator
HCI command packet	0x01
HCI ACL data packet	0x02
HCI SCO data packet	0x03
HCI event packet	0x04

Table A1. HCI packets

The HCI packet indicator shall be sent immediately before the HCI packet. When the entire HCI packet has been received a new indicator should be expected.

The default speed setting is 57.6 kb/s and can be changed by sending a specified bit stream to the I/F - see Appendix C on how to change the speed setting of the UART.

When the speed set-up for the UART is made for both Host_ A & B, the Command Packets can be sent and Event Packets received by the hosts. See HCI over UART in part H:4 of the Bluetooth System v1.0B for detailed information regarding parameters and protocol.

Soft Reset

First HCI command packet to send should be the RESET packet.

A Command_Complete_Event with a status parameter should be returned to the host.

Buffer information
Buffer information should be exchanged between the module and respective host by using HCI commands.

- Read_Buffer_Size: Providing the host with information on buffer size for ACL and SCO data packets for the module returned with a Command_Complete_Event packet. The host shall use this information for controlling the transmission
- Host_Buffer_Size: Providing the module with information on buffer size for ACL and SCO packets to the host.

It is the host that manages the data buffers of the Host Controller on the module.

Timers
It could be necessary to set important timers used by the module for time out handling. The timers are all set by writing to registers using HCI commands.

The default values can be checked in Specification of the Bluetooth System v1.0B part H:1 or by using Read_xxx_xxx commands.

Bluetooth Address
The hosts, using the HCI command Read_BD_ADDR will find the Bluetooth address of the module by the Command_Complete_Event with the BD_ADDR as a parameter.

By Remote_Name_Request, the BD_ADDR of the remote module can also be found.

Inquiry
The HCI command Inquiry with the parameters LAP, Inquiry_Length, and Num_Responses can also be used for collecting BD_ADDR of remote Bluetooth units.

Creating a Point-to-point connection

Page Scan mode
To set a Bluetooth module in the mode for being able to connect to (Host_B), page scan mode, there are some settings that should be performed.

Command	OCF	Command parameters	Return parameters
HCI_Write_Scan_Enable	0x001A	Scan_Enable	Status

Table A2. HCI Write Scan Enable OCF code

Furthermore the setting of authentication and encryption should be disabled (default) by using the:
- Write_Authentication_Enable
- Write_Encryption_Mode

The basic settings for getting into scan mode could be according to the below suggested script list.
- Read Buffer Size
- Set Event Filter
- Write Scan Enable: (Scan Enable: 0x03)
- Write Voice Setting: (Voice Channel Setting: 0x0060)
- Write Authentication Enable: (Authentication Enable: 0x00)
- Set Event Filter: (Connection Setup Filter: Connections from All Devices, Auto Accept: 0x02)
- Write Connection Accept Timeout: (Connection Accept Timeout: 0x2000)
- Write Page Timeout: (Page Timeout: 0x3000)

Page mode
The Create_Connection command is used to set-up a link to another Bluetooth device.

Command	OCF	Command parameters	Return parameters
HCI_Create_Connection	0x0005	BD_ADDR Packet_Type Packet_Scan_Repetition_Mode Packet_Scan_Mode Clock_Offset Allow_Role_Switch	

Table A3. HCI Create Connection OCF code

Create_Connection:

BD_ADDR: 0xYYYYYYYYYYYY,
Packet Type: 0x0008,
Page Scan Repetition Mode: 0x01,
Page Scan Mode: 0x00,
Clock Offset: 0x0000
Allow_Role_Switch: 0x00

This command will cause the Link Manager to try to create a connection to the Bluetooth module with the appropriate BD_ADDR. The local Bluetooth module (Host_A) starts the paging process to set up a link to the page-scanning remote device (Host_B).

By LMP the negotiation between the two Bluetooth modules Link Managers (LM) the link set-up can be completed.

ACL link up and running
Host_A is the master of the point-to-point piconet and Host_B is slave. The unit starting the paging process is by definition the master. The link set-up is completed when the event
Connection_Complete_Event is returned to both Host_A and Host_B with the connection handle as one of the parameters and the status parameter 0x00 (success). When Bluetooth link is up and running the HCI data packets can be sent from host to host.

The host must take care of generating the packages going from Host to Host Controller in the module over the UART I/F, in the same way the Host must arrange the packages received from the Host Controller.

Both sides need information on what kind of data is received, to be able to interpret the bit flow correctly.

For extensive information on setting up a Bluetooth link please refer to Bluetooth System v1.0B Appendix IX Message Sequence Charts

Adding an SCO link
When creating a voice connection using the PCM I/F, an ACL link must be up and running between the two devices, an SCO link can thereafter be added.

The control of the PCM I/F (FS, PCM_IN/OUT and PCM_CLK) is handled by Ericsson specific HCI command – see Appendix C.

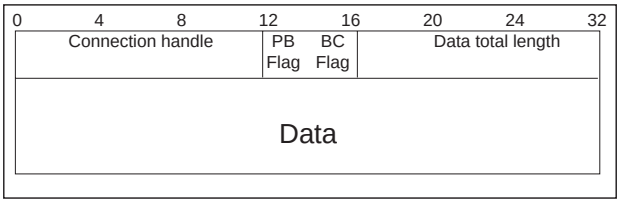


Table A4. ACL data packet

Driving SW

HCI- API
The Bluetooth module includes all HCI command capabilities according to Bluetooth System v1.0B. Furthermore, there is some Ericsson specific commands available for accessing HW registers and HW control – see Appendix C. SW for driving the module should be developed for the HCI interface.

There is source code SW available with Ericsson Bluetooth Developers Kit (EBDK), see Appendix B, which can be used for driving the module. This SW comes with the EBDK and has an HCI-API for application development on the HCI I/F.

Higher layer-API
Software (HCIdriver, L2CAP, RFCOMM and SDP) are available in a generic, source code format, i.e. to be adapted to various operating systems.

- HCIdriver – implements the HCI command driver used by the host
- L2CAP – handles protocol multiplexing, segmentation and re-assembly of packets
- RFCOMM – provides a serial port emulation over the L2CAP protocol
- SDP – Service Discovery Protocol provides information on the services available on a Bluetooth device

Additional SW for the application shall be developed for the actual application on top of the RFCOMM API. If the application is according to a SIG predefined profile, it should be implemented accordingly. New applications can be the driver of the specification of new profiles decided by the SIG – see Specification of the Bluetooth System v1.0B Profiles.

APPENDIX B

Development tools

Bluetooth Development Kit (EBDK)
The easiest way of getting started is to use the Bluetooth Developers Kit. It provides all parts necessary for developing applications for the Bluetooth module.

Available are:

- PC plug&play
- Demos using radio/baseband
- Macro capability
- C++ v5.0 Source code for use in applications
- HCIdriver, L2CAP, SDP and RFCOMM for applications using UART communication
- Pins for electrical measurements
- Antennas

Development can easily take place on the EBDK platform and thereafter the implementation of the full Bluetooth capability can be setup by developed SW/HW and the Bluetooth module.

Software (HCIdriver, L2CAP, RFCOMM and SDP) will be available in source code for PC.

- HCIdriver – implements the HCI command driver used by the host
- L2CAP – handles protocol multiplexing, segmentation and re-assembly of packets
- RFCOMM – provides a serial port emulation over the L2CAP protocol
- SDP – Service Discovery Protocol provides information on the services available on a Bluetooth device

Technical support is available from the EBDK distributor. Please contact Ericsson Microelectronics for ordering and information regarding the EBDK and regarding extra daughter board with ROK 101 008 as add-on to the EBDK.

Bluetooth Starter Kit (EBSK)
A very small convenient kit, which preferably is used in point-to-multipoint configuration designs, based on the Bluetooth module ROK 101 008.

Please contact Ericsson Microelectronics for ordering and information regarding the Ericsson Bluetooth Starter Kit.

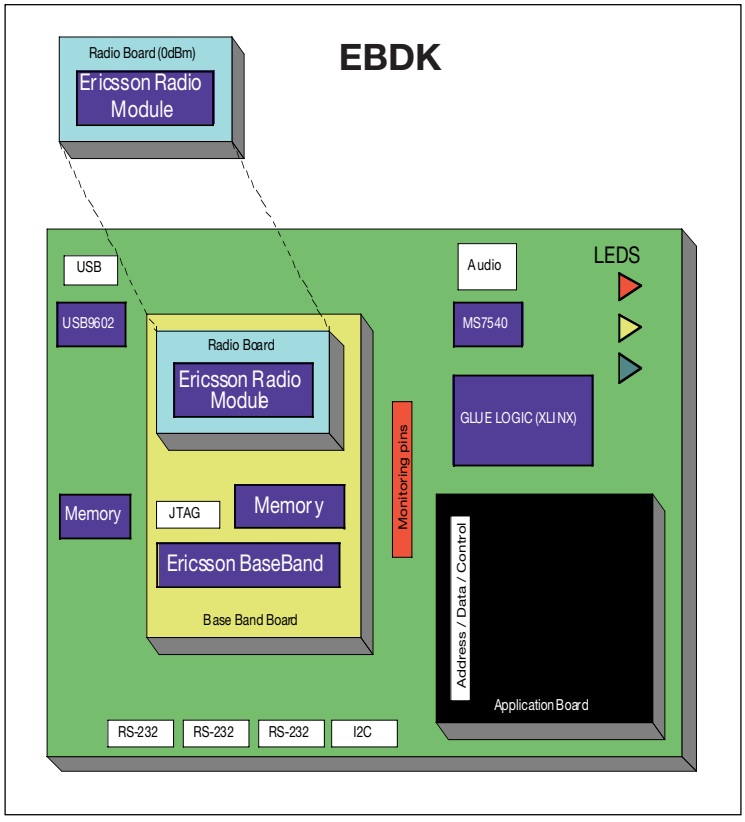


Figure B1. Bluetooth Development Kit (EBDK)

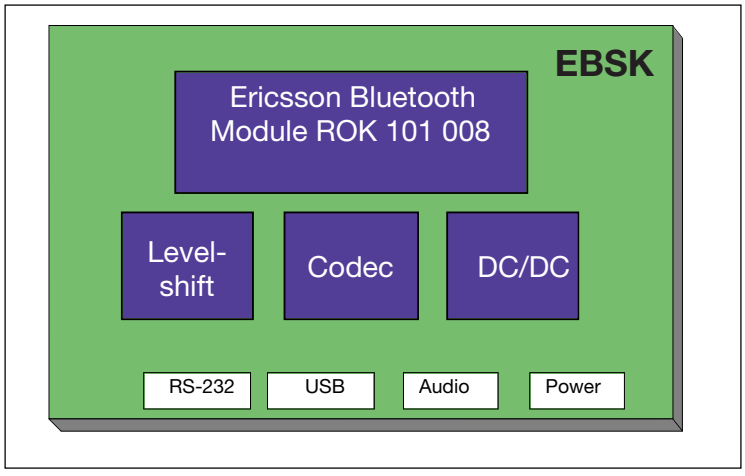


Figure B2. Bluetooth Starter Kit (EBSK)

APPENDIX C

Bluetooth Information

This part will be updated and new information will be added continuously.

Radio

Fast Frequency hopping (1600channel hop/s) with 79(23) channels available (2.402 to 2.480 GHz) and a symbol rate of 1Ms/s over the air exploits the maximum channel bandwidth in the unlicensed ISM band.

To sustain a high transfer rate in busy radio environment, the frequency hopping together with advanced coding techniques maximizes the throughput.

During Page and Inquiry the hopping frequency is risen to 3200 hops/s to enhance the time needed for connection set-up.

Modulation technique is a binary Gaussian Frequency Shift Keying GFSK, with a BT product of 0.5. The channel bandwidth is 1 MHz and the frequency deviation from the carrier frequency of the RF channel is between +/-140 to +/-175 kHz for representing a '1'/'0'.

A rapid process is ongoing to harmonise Spanish, French and Japanese frequency ranges with the rest of the world.

Data and parameter formats

There are exceptions in the Bluetooth system for data and parameter formats – general rules below.

- All values are in Binary and Hexadecimal little Endian formats
- Negative values must use 2's complement format
- Array parameter notation is parameterA[i], parameterB[i],…
- All parameter values are sent/received in little Endian format. The least significant byte is sent first – unless noted otherwise.

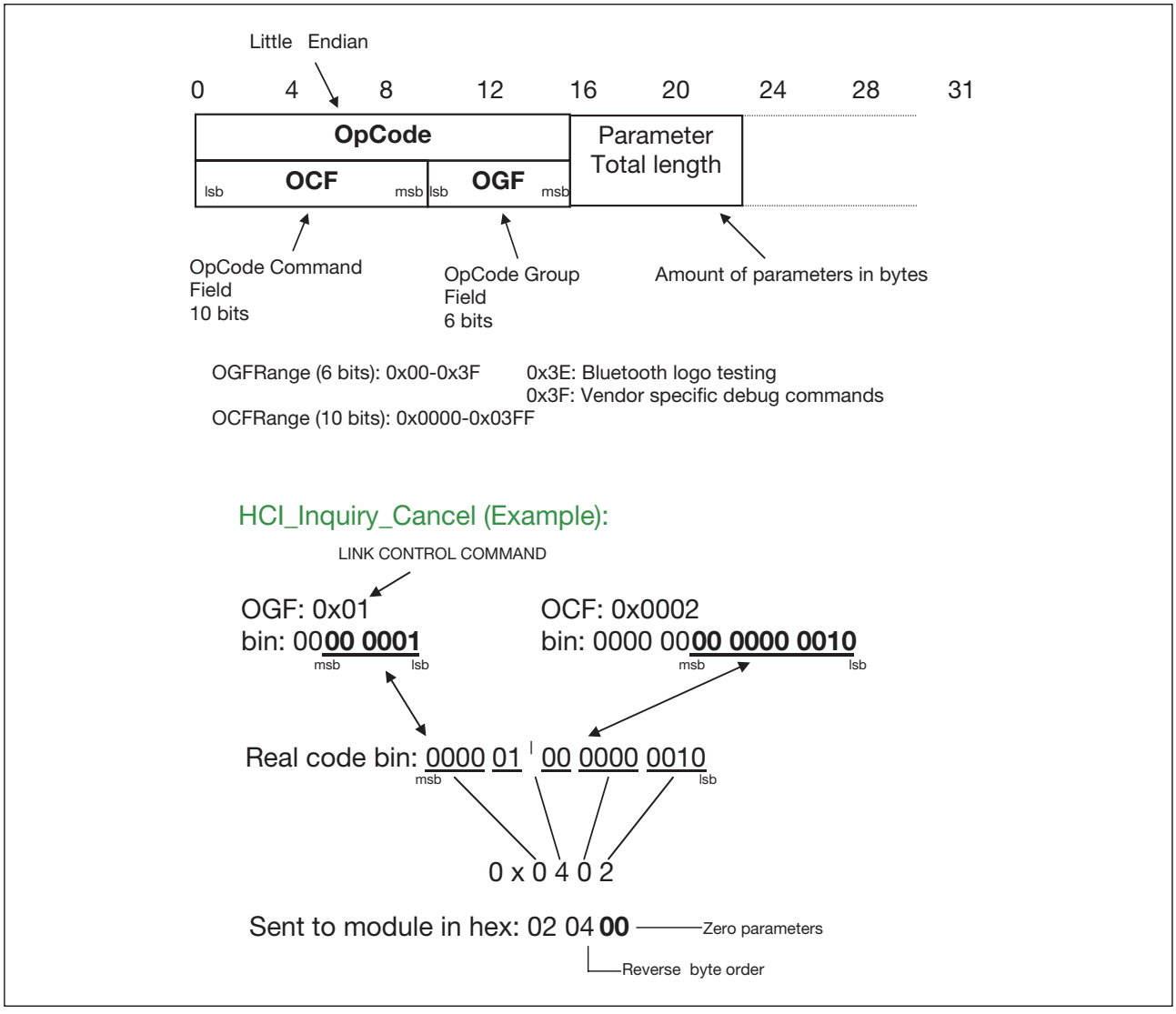


Figure C1. Byte order sent to module

HCI Opcodes
The Opcodes have been changed during the ongoing standardisation work. Below is the description on how to send opcodes to the module.

Below is the general HCI command packet format depicted as well as a byte order description.

UART speed setting
The baud rate is changed with an Ericsson specific HCI command.
HCI_Ericsson_Set_Uart_Baud_Rate

The command has one parameter, baud rate - one byte long according to the table below . The op-code for the command is 0xfc09 - the last figure is due to a possible change.

Sending the command should be performed accordingly: 09 fc 01 yz , where yz is the chosen baud rate from the table.

UART speed	Parameter to send
460.8 kbps	00000
230.4 kbps	00001
115.2 kbps	00010
57.6 kbps	00011
28.8 kbps	00100
14.4 kbps	00101
7200 bps	00110
3600 bps	00111
1800 bps	01000
900 bps	01001
153.6 kbps	10000
76.8 kbps	10001
38.4 kbps	10010
19.2 kbps	10011
9600 bps	10100
4800 bps	10101
2400 bps	10110
1200 bps	10111
600 bps	11000
300 bps	11001

Table C2. UART speed setting parameter

The op-code is sent in reverse byte order. 01 is the parameter length, in this case one byte. Remember to add the HCI packet indicator.

Observe - When changing the baud rate for the module the host also has to change the baud rate.

Observe - Removing power to the module the baud rate will be reset to 57.6 kbps.

Observe -Two zeros are not printed in the beginning of the binary parameters below. The length of the parameter is 1 byte.

Ericsson specific HCI commands
By using the Ericsson specific HCI command there are a number of features available for the application design.

Contact Ericsson Microelectronics for a command reference list.

Antennas
Antenna design is not specified and standardised in the Bluetooth System v1.0B.

Many different types of antennas can and will be used. Application specific antennas suitable for production are expected to be a new market for antenna designs.

Ericsson Microelectronics have antennas for the EBDK and other development kits. Contact Ericsson Microelectronics for information on antennas.

Contacting Ericsson Microelectronics
For further information regarding Bluetooth technology, components and development tools, please contact Ericsson Microelectronics:

Country	Frequency range	RF channels	
Europe & USA	2400-2483.5 MHz	f = 2402 + k MHz	k = 0....78
Japan	2471-2497 MHz	f = 2473 + k MHz	k = 0....22
Spain	2445-2475 MHz	f = 2449 + k MHz	k = 0....22
France	2446.5-2483.5 MHz	f = 2454 + k MHz	k = 0....22

Table C1. Frequency ranges used.

Ericsson Microelectronics
SE-164 81 Kista, Sweden
+46 8 757 50 00
www.ericsson.com/microelectronics

For local sales contacts, please refer to our website
or call: Int + 46 8 757 47 00, Fax: +46 8 757 47 76

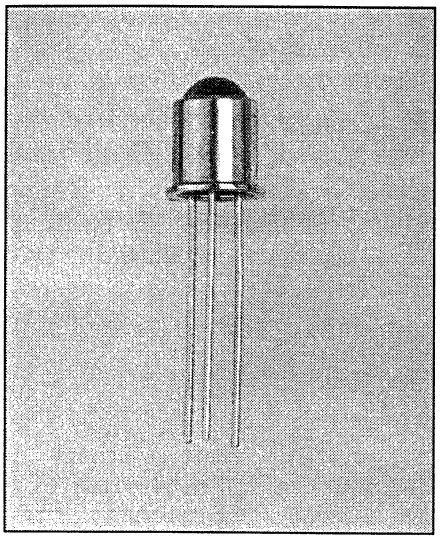
Data Sheet

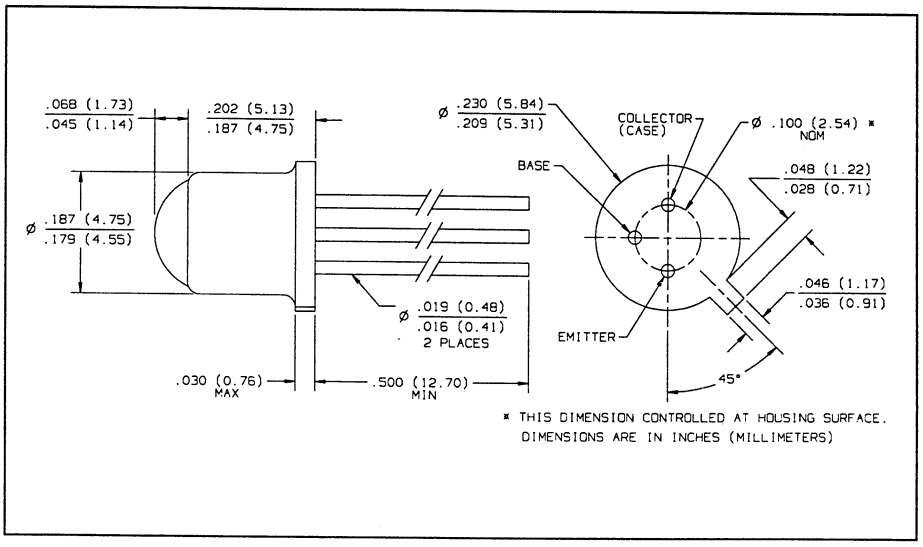
EN/LZT 146 106 R1A

Appendix F
Optek OP805 Phototransistor
Data Sheet

NPN Silicon Phototransistors

Types OP800SL, OP801SL, OP802SL, OP803SL, OP804SL, OP805SL





Features

- Narrow receiving angle
- Variety of sensitivity ranges
- Enhanced temperature range
- TO-18 hermetically sealed package
- Mechanically and spectrally matched to the OP130 and OP231 series of infrared emitting diodes
- TX/TXV processing available

Description

The OP800SL series device consists of an NPN silicon phototransistor mounted in a hermetically sealed package. The narrow receiving angle provides excellent on-axis coupling. TO-18 packages offer high power dissipation and superior hostile environment operation. The base lead is bonded to enable conventional transistor biasing.

Replaces

OP800 and K5251 series

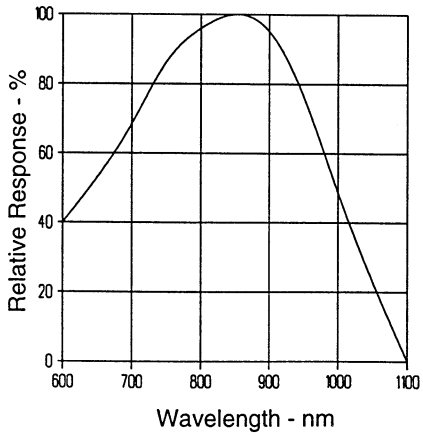
Absolute Maximum Ratings (T_A = 25° C unless otherwise noted)

Collector-Base Voltage	30 V
Collector-Emitter Voltage	30 V
Emitter-Base Voltage	5.0 V
Emitter-Collector Voltage	5.0 V
Continuous Collector Current	50 mA
Storage Temperature Range	-65° C +150° C
Operating Temperature Range	-65° C +125° C
Lead Soldering Temperature [1/16 inch (1.6 mm) from case for 5 sec. with soldering iron]	260° C(1)
Power Dissipation	250 mW(2)

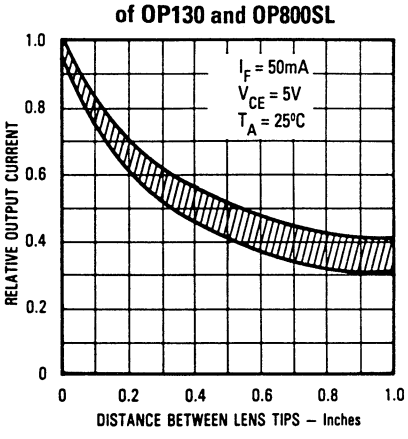
Notes:
(1) RMA flux is recommended. Duration can be extended to 10 sec. max. when flow soldering.
(2) Derate linearly 2.5 mW/° C above 25° C
(3) Junction temperature maintained at 25° C
(4) Light source is an unfiltered tungsten bulb operating at CT = 2870 K or equivalent infrared source.

Typical Performance Curves

Typical Spectral Response



Coupling Characteristics of OP130 and OP800SL



Types OP800SL thru OP805SL

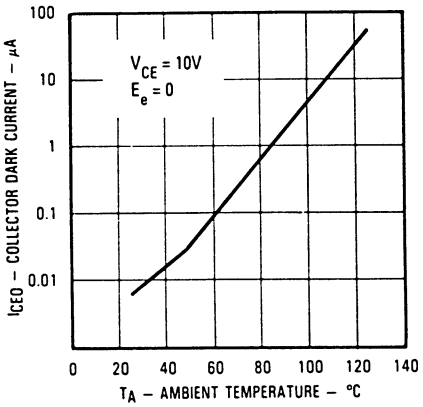
Electrical Characteristics (T_A = 25° C unless otherwise noted)

SYMBOL	PARAMETER		MIN	TYP	MAX	UNITS	TEST CONDITIONS
$I_{C(ON)}$	On-State Collector Current	OP800SL	0.5			mA	V_{CE} = 5 V, E_e = 5 mW/cm^2(3)(4)
		OP801SL	0.5		3.0	mA	
		OP802SL	2.0		5.0	mA	
		OP803SL	4.0		8.0	mA	
		OP804SL	7.0		22.0	mA	
		OP805SL	15.0			mA	
I_{CEO}	Collector Dark Current				100	nA	V_{CE} = 10 V, E_e = 0
$V_{(BR)CEO}$	Collector-Emitter Breakdown Voltage		30			V	I_C = 100 µA
$V_{(BR)CBO}$	Collector-Base Breakdown Voltage		30			V	I_C = 100 µA
$V_{(BR)ECO}$	Emitter-Collector Breakdown Voltage		5.0			V	I_E = 100 µA
$V_{(BR)EBO}$	Emitter-Base Breakdown Voltage		5.0			V	I_E = 100 µA
$V_{CE(SAT)}$	Collector-Emitter Saturation Voltage				0.40	V	I_C = 0.4 mA, E_e = 5 mW/cm^2(4)
t_r	Rise Time			7.0		µs	V_{CC} = 5 V, I_C = 0.80 mA,
t_f	Fall Time			7.0		µs	R_L = 100 Ω, See Test Circuit

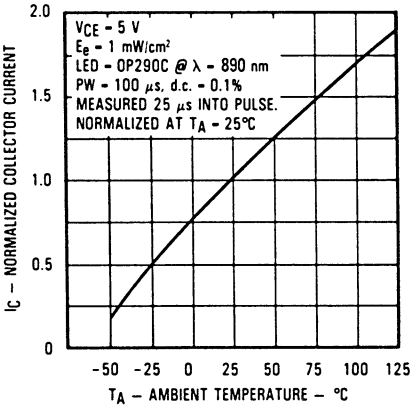
PHOTOSENSORS

Typical Performance Curves

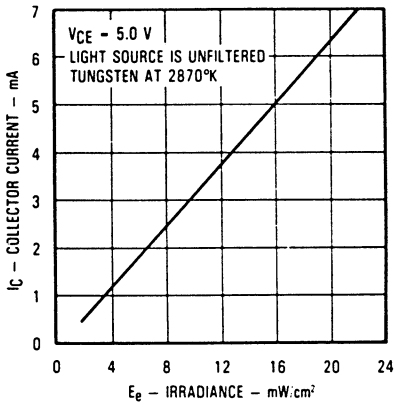
Collector Dark Current vs. Ambient Temperature



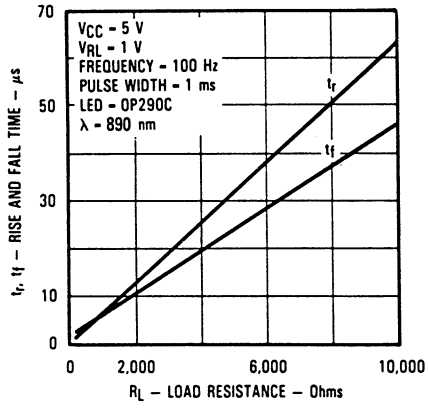
Normalized Collector Current vs. Ambient Temperature



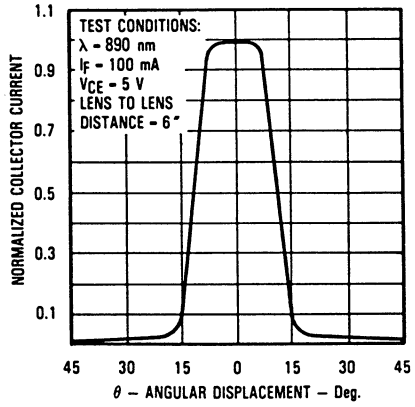
Collector Current vs. Irradiance



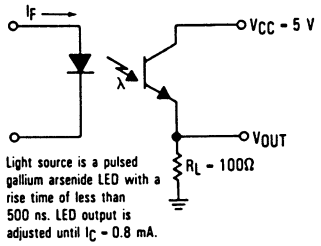
Rise and Fall Time vs. Load Resistance



Normalized Collector Current vs. Angular Displacement



Switching Time Test Circuit



Optek Technology, Inc. 1215 W. Crosby Road Carrollton, Texas 75006 (972)323-2200 Fax (972)323-2396

References

[1] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing.* Second Edition. Upper Saddle River: Prentice-Hall, Inc., 2001.

[2] S. O. Kasap, *Optoelectronics and Photonics: Principles and Practices.* First Edition. Upper Saddle River: Prentice-Hall, Inc., 2001.

[3] Paul Horowitz and Winfield Hill, *The Art of Electronics.* Second Edition. Cambridge: Cambridge University Press, 1989.

[4] Jennifer Bray and Charles F. Sturman, *Bluetooth™ Connect Without Cables.* First Edition. Upper Saddle River: Prentice-Hall, Inc., 2001.

[5] David Kammer, Gordon McNutt, Brian Senese and Jennifer Bray, *Bluetooth Application Developer's Guide: The Short Range Interconnect Solution.* Rockland: Syngress Publishing, Inc., 2002.

[6] EG&G Optoelectronics, *What are Phototransistors?.*
http://www.engr.udayton.edu/faculty/jloomis/ece445/topics/egginc/pt_def.html,
EG&G Inc., 1997

[7] Eric Seale , *Phototransistors.*
http://encyclobeamia.solarbotics.net/articles/phototransistor.html
February 26, 2002

[8] *Silicon Photovoltaic Cells, Photodiodes and Phototransistors*.
http://www.powerpulse.net/powerpulse/archive/aa_050701a1.stm
Darnell Group Inc. 2001

[9] *Specification of the Bluetooth System: Core.* Version 1.1, Bluetooth SIG, Inc., 2001

[10] *PIC16F87X, 28/40-Pin 8-Bit CMOS FLASH Microcontrollers.* 30292c, Product Data Sheet, Microchip Technology Incorporated, 2001.

[11] *PICMicro™ Mid-Range MCU Family Reference Manual.* 33023a,
Product Data Sheet, Microchip Technology Incorporated, 1997.

[12] *ROK 101 008 Bluetooth Module,* rok101008_146106r1a, Product Data Sheet,
Kista Sweden: Ericsson Microelectronics, 2001

[13] *MM74HC4051 • MM74HC4052 • MM74HC4053 8 • Channel Analog Dual 4-Channel Analog Multiplexer • Triple 2-Channel Analog Multiplexer*. DS005353.prf,
© Fairchild Semiconductor Corporation, 1999

Vita

NAME: Christopher T. Harrower

PLACE OF BIRTH: Moose Jaw, Saskatchewan

YEAR OF BIRTH: 1978

SECONDARY EDUCATION: Murdoch MacKay Collegiate Institute, 1996

NAME: Richard J. Rodd

PLACE OF BIRTH: Hamilton, Ontario

YEAR OF BIRTH: 1977

SECONDARY EDUCATION: J. H. Bruns Collegiate, 1995

NAME: Julian P. Z. Nedohin-Macek

PLACE OF BIRTH: Winnipeg, Manitoba

YEAR OF BIRTH: 1976

SECONDARY EDUCATION: Elliot Lake Secondary School, 1995

NAME: R. Serge LeBlanc

PLACE OF BIRTH: Montréal, Québec

YEAR OF BIRTH: 1967

POST-SECONDARY EDUCATION: Computer Engineering Technology,
Red River College, 1994