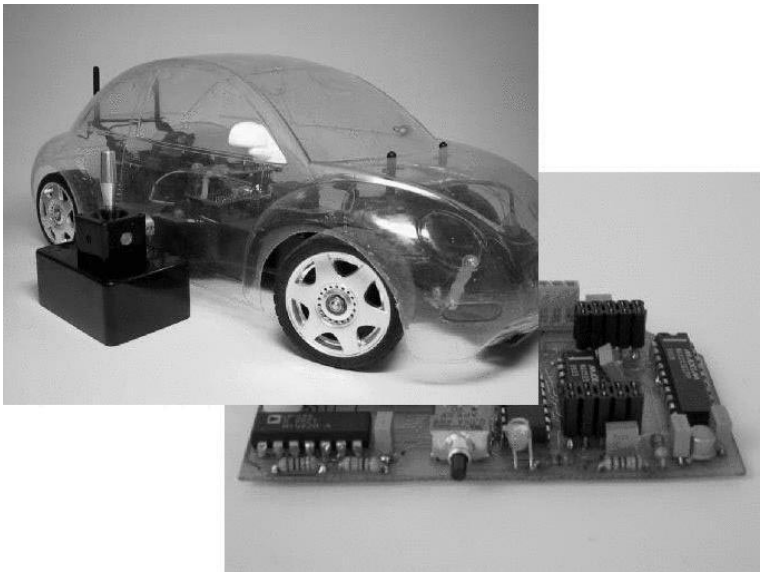
ISSN 0280–5316
ISRN LUTFD2/TFRT--5652--SE

Implementing a Wireless I/O Unit using Bluetooth

Per Nilsson
Johan Brodin



Department of Automatic Control
Lund Institute of Technology
November 2000

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
November 2000

Document Number
ISRN LUTFD2/TFRT--5652--SE

Author(s)
Johan Brodin
Per Nilsson

Supervisor
Bo Bernhardsson (LTH)
Lennart Andersson (Sigma Exallon)

Sponsoring organisation

Title and subtitle
Implementing a Wireless I/O Unit using Bluetooth
(Implementering av trådlös in- och utenhet med Bluetooth)

Abstract

Bluetooth is a new standard for wireless communication. The aim so far has mostly been to use this technology in an office environment. This master thesis considers the advantages of Bluetooth in an industrial environment.

A general circuit board that uses Bluetooth for wireless communication has been constructed. The board can be seen in the picture to the right. This platform can easily be connected to almost any electrical device, which then gains the benefits of wireless communication. The devices can for example be carports, door locks, or sensors and actuators in industrial processes. The developed software does not support any Bluetooth profile, but is general and easy to expand.

To show the possibilities with the hardware platform a 1/10th-scale car application has been developed. The car is steered by a joystick and both devices are attached to the general hardware platform.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
154

Recipient's notes

Security classification

Table of Contents

Acknowledgements

There were a lot of unanswered questions when the master thesis project started. It was not clear at all how to construct the wireless I/O unit using Bluetooth. The success of the master thesis project is much due to the combination of knowledge we have. We are both masters of science students but in two different fields, computer science and electrical engineering. The project could not have been done in 20 weeks without our different experience and knowledge.

Our supervisors, Ph.D. Lennart Andersson at Sigma Exallon System and Professor Bo Bernhardsson at the Department of Automatic Control, have both contributed to the final result. Lennart Andersson by helping us in the daily work and Bo Bernhardsson with the master thesis paper.

People who have contributed to the master thesis project with lots of ideas and enthusiasm are Henrik Svensson, Ph.D Johan Eker and Bo Lincoln.

We would like to thank Rolf Braun at the Department of Automatic Control for helping us etch the circuit board and Lars Andersson for his contribution of knowledge about radio cars.

We would also like to thank Sigma Exallon System, especially Jan Enejder and Anders Ohlsson, for providing us with all resources we have needed.

Finally, are we very thankful that our girlfriends, Sara and Ulrika, have put up with us during the project.

Abstract

Bluetooth is a new standard for wireless communication. The aim so far has mostly been to use this technology in an office environment. This master thesis considers the advantages of Bluetooth in an industrial environment.

A general circuit board that uses Bluetooth for wireless communication has been constructed. This platform can easily be connected to almost any electrical device, which then gains the benefits of wireless communication. The devices can for example be carports, door locks, or sensors and actuators in industrial processes. The developed software does not support any Bluetooth profile, but is general and easy to expand.

To show the possibilities with the hardware platform a $1/10^{th}$-scale car application has been developed. The car is steered by a joystick and both devices are attached to the general hardware platform.

1. Introduction

Bluetooth is the fastest growing industrial standard in the history. It is a wireless communication technology, which can transmit both speech and data at a relatively high speed. For a detailed description of Bluetooth, see Appendix F. Bluetooth is a low cost solution to replace cables in an office environment. The technology is supported by 2000 companies around the world, and this will ensure that a wide range of different devices will be compatible which each other. The market potential for Bluetooth is estimated to be more than one billion units in a couple of years.

The main goal with the master thesis project is to consider the advantages of Bluetooth in an industrial environment. Questions we have investigated are: How should a general Bluetooth I/O unit be constructed? What functionality is required for the I/O unit? How small can the delay and sample time be made? Is it possible to steer retransmission of lost packages? How should lost connections during execution be handled?

This master thesis project has been developed at the company Sigma Exallon Systems AB, and at the Department of Automatic Control at Lund University. The department's goal has been to develop a generic Bluetooth communication solution, and Sigma's desire has been to implement it in a prototype that shows the potential of Bluetooth. Constructing a generic I/O unit called Bluetooth Control Card (BCC), using Bluetooth for wireless communication has fulfilled this goal.

The communication solution can be seen in Figure 1.1. Analog signals at the BCC's inputs are sampled and transmitted over the Bluetooth channel to the PC. At the same time signals can be sent to the BCC and set at the outputs. To speed up the development process a Bluetooth module from Ericsson was used instead of integrating a module into the BCC. The BCC can be attached to many devices, e.g. door locks, carports, and sensors and actuators in industrial processes.

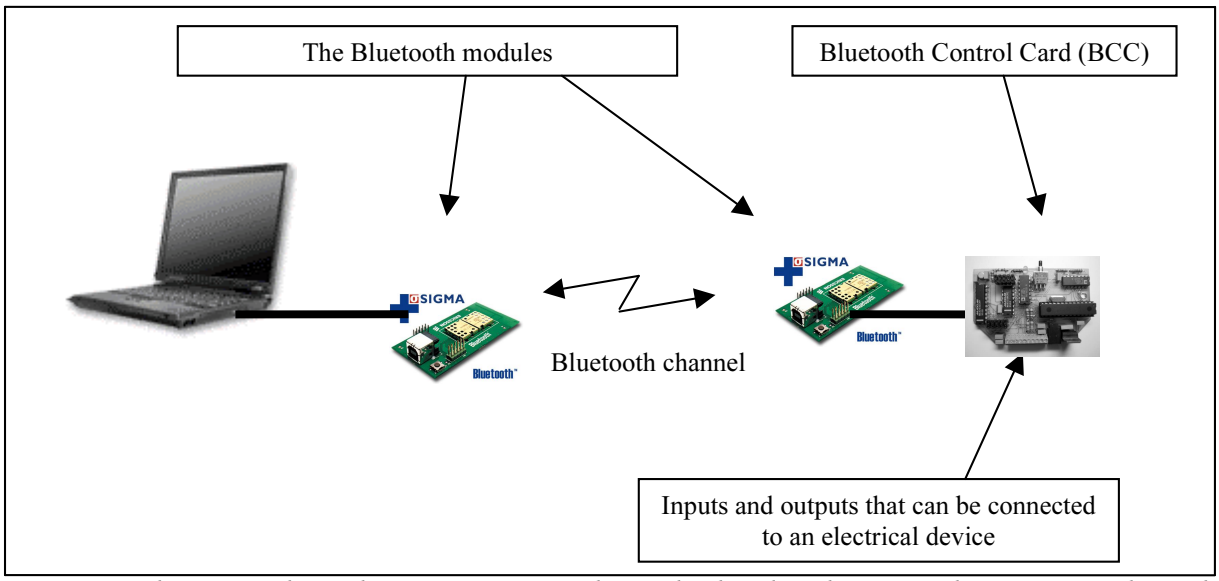


Figure 1.1: The Figure shows the communication solution developed in the master thesis project. Bluetooth modules are attached to the PC and to the Bluetooth Control Card (BCC). Data packages can be sent in both directions over the Bluetooth channel. The BCC is a circuit board developed in the master thesis project and it contains inputs and outputs. The inputs are sampled at a user defined period and sent over the Bluetooth channel. The PC sets the outputs on the BCC.

The master thesis report is organized as follows.

In Section 2 the hardware solution is discussed. Section 3 shows the software solution and the definition of the master slave concept. Section 4 defines the communication protocol between the master and slave. Section 5 describes the radio car application. Section 6 illustrates a general application running on the PC, controlled by the BCC. Conclusions are drawn in Section 7. Section 8 contains the references.

Several appendices are included in the report. The time planning and all documentation for the construction of hardware and software are incorporated. A general appendix of Bluetooth is attached to give an overview of the technology. Appended is also the press release from the Bluetooth Developers Conference in San José [4], where the master thesis project will be demonstrated. A specific appendix on the performance and characteristic of the BCC and a user's manual on the Beetle radio car application are also included.

2. Hardware Solution

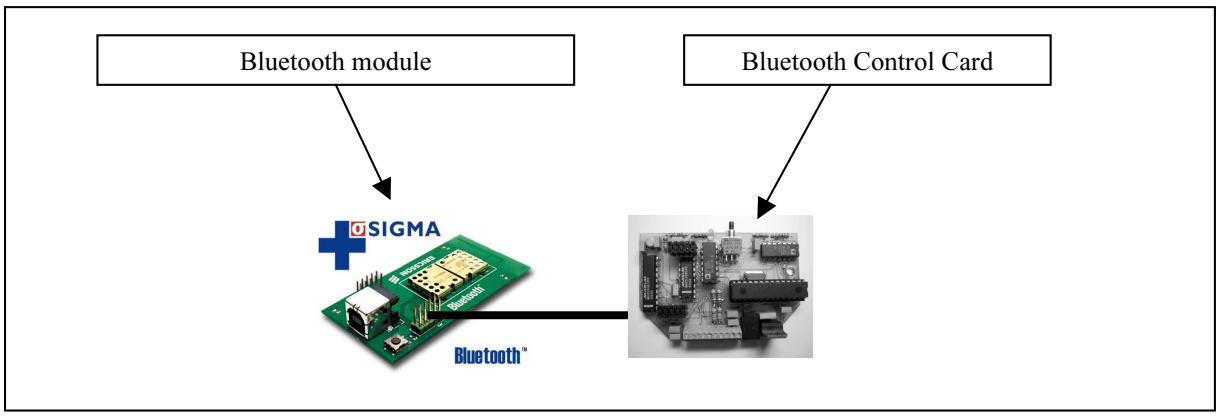


Figure 2.1: The two circuit boards that the hardware solution consists of. On the board to the right is the Bluetooth chip and the antenna. The board to the right is constructed in the master thesis project and it contains a micro controller that controls the inputs and outputs. Figure 2.2 below shows an overview of the different components at the BCC.

The hardware solution consists of two circuit boards, see Figure 2.1. The board to the left is the Bluetooth module and antenna. This board is constructed by Ericsson and is sold as an Application and Training Tool Kit. The board to the right is the one that is constructed in the master thesis project and is called Bluetooth Control Card (BCC). On this board the Bluetooth module and the antenna could have been implemented, but it would then have taken longer time to construct the hardware. This would of course have been done in a commercial product to make it cheaper and smaller.

The BCC has all the logic for the analog inputs and outputs. Below is described how it is constructed and how the different components on the card interact. There is also a Section that describes how to access and use the card.

2.1 Circuit Board

The task of the BCC is to be the communication link between the Bluetooth module and the inputs and outputs. The data it receives from the Bluetooth module is forwarded to the outputs. The values that the inputs have is sampled by the card and sent to the Bluetooth module. A micro controller handles all the communication to and from the circuit board. Figure 2.2 below shows an overview of the different components of the BCC. To make the drawing easier to study some lines have been made thicker. The thicker lines symbolize that two or more wires are implemented equally. For example do the four inputs go to one operational amplifier each and then four wires go to the micro controller.

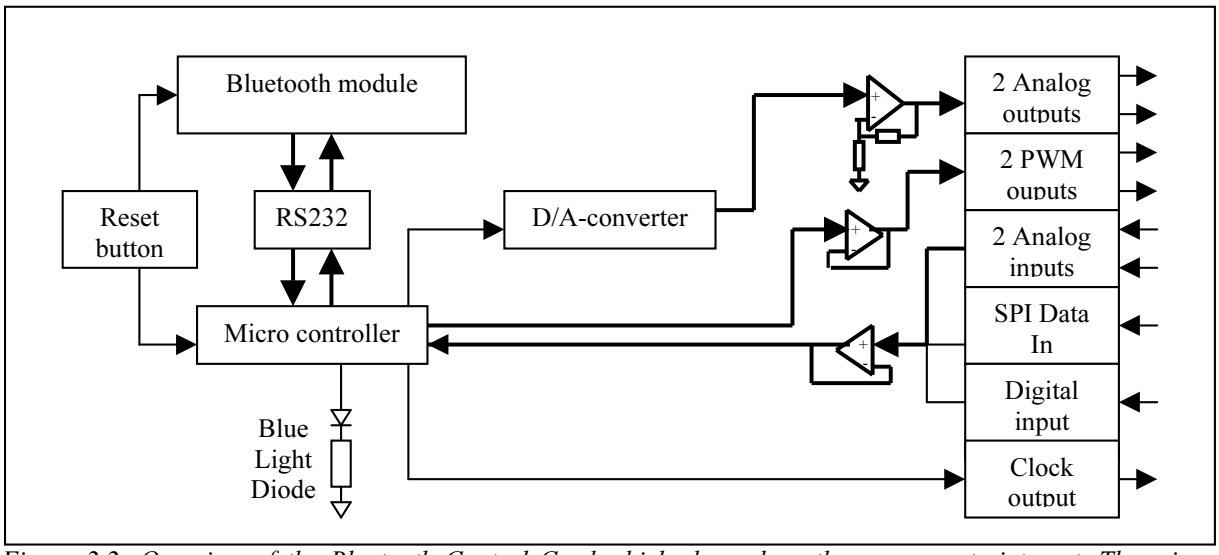


Figure 2.2: Overview of the Bluetooth Control Card which shows how the components interact. The micro controller's tasks are to receive information for the outputs from the Bluetooth module and set this on the outputs. It also samples the inputs and sends this information to the Bluetooth module. To see the schematic view which shows all the connections on the BCC see Figure 2.3.

The chosen micro controller is a PIC16F876. It is a good choice since it has a lot of built in functionality such as A/D-converters, PWM modules, timers, flash memory, an SPI module and a USART. This makes both the hardware and software easier to implement. The A/D-converters do not have to be asserted on the board and the software for timing issues is easy to implement because of the already implemented timers. The communication between the micro controller and the Bluetooth module is done over the serial channel RS232. The RS232-component converts the micro controller's voltages to RS232 voltages. The maximum speed over the RS232 channel is constrained by the micro controller to 57.6 kbps.

The outputs from the card are two analog outputs and two PWM (Pulse Width Modulation) outputs. The micro controller sends the digital values to a D/A-converter. The D/A-converter delivers a value between 0 and 2.4V, which is amplified a factor 2 to 0 to 5V by an operational amplifier. The PWM period and duty cycle is received from the remote Bluetooth device and the correct pulse is created in the micro controller and sent to an operational amplifier. The amplifier has unity gain, but is needed to drive the outputs, as the micro controller can not drive any outputs itself. For further information about the PWM signals see Section 4.1.

The inputs consist of two analog inputs and an SPI (Serial Peripheral Interface). All the signals from the inputs go to operational amplifiers with unity gain. This is to protect the inputs at the micro controller, as the highest voltage that can come from the amplifier is 5V and the lowest is 0V. The voltages from the analog inputs go to the A/D-converters in the micro controller.

The SPI consists of a digital input, an SPI Data In (SDI), and one clock output from the micro controller. The digital input decides when the BCC can receive data on the SDI. The device that sends the data uses the clock from the micro controller. The micro controller can not simultaneously set the analog outputs and sample the SDI. For further information about the SPI see Section 4.2.

When the reset button is pressed it resets both the BCC's micro controller and the Bluetooth module's micro controller. There is also a blue light diode on the board. This is controlled by the micro controller and is used to show when the reset button is pressed, when a connection with another Bluetooth device is established, and when an software error occur.

Figure 2.3 shows a schematic view of the BCC with all the components. The power supply of the board is 5.5 to 35V and this voltage is stabilized by a 1uF capacitor. The internal power supply then becomes 5V from the voltage regulator. The power supply to the Bluetooth module is also 5V. To make the board less sensitive to disturbances, capacitors of 100nF are put at each component, and to stabilize the signals on the inputs, capacitors with the same capacitance are put there. The two components "Bluetooth Signals" and "Bluetooth Power Supply" at the bottom of the figure are the sockets that are connected to the Bluetooth module. The component to the right has the pins that it is possible to connect to.

It was a little bit tricky to make the reset button able to reset both cards. Both cards should have zero volts when they are reset, but otherwise the BCC's reset pin should usually have 5V and the Bluetooth module should not be connected at all. To solve this, a button with three connections is used as can be seen above the micro controller's clock. When the reset button is pressed all three wires are connected to each other.

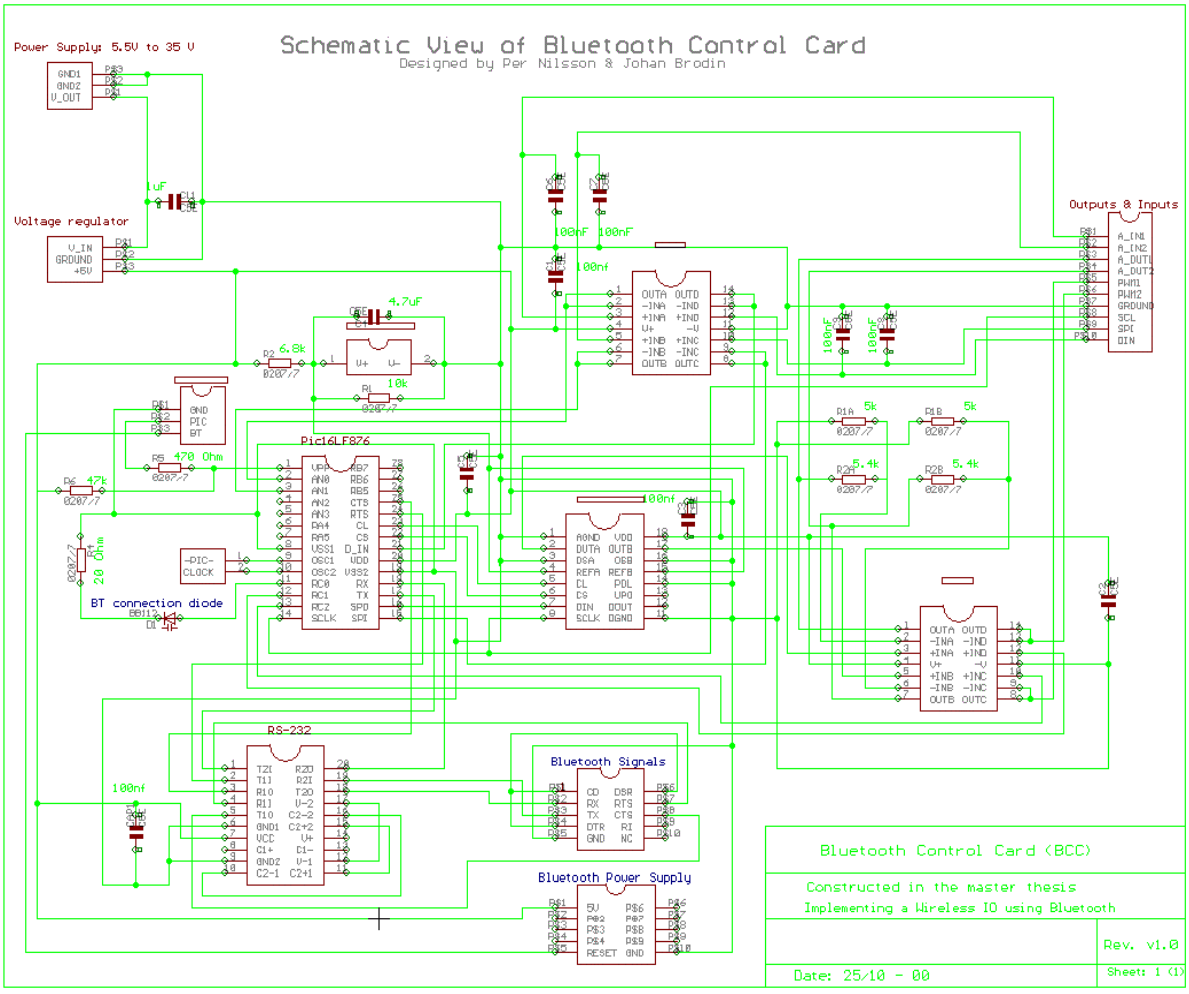


Figure 2.3: This is the schematic view of the BCC which shows how all the wires are drawn. The drawing was made in a Eagle CAD-program. For further information about the development environment see Section 2.4.

During the development of the BCC an iterative approach was chosen and two boards were built. The first board was built to test and study the construction so the second and final board could be implemented correctly. The first BCC that was constructed is 8*10 cm and has two large expensive DC/DC converters. The idea was to get a working board as soon as possible in order to start developing the software. The second and final board is 8*5 cm and is connected back to back to the Bluetooth module.

The first board can be seen in Figure 2.4. The board is driven by both 3.3V and 5V. The reason behind the different power supplies was that the Bluetooth chip is driven by 3.3V, and the outputs were driven with 5V. This solution could not be used, since the Bluetooth module from Ericsson is driven with 5V instead of 3.3V. The first hardware design did work despite the design error. The construction of the first board was good for the master thesis project, since a prototype became available early in the development. Although it had drawbacks both in size and cost it was perfect as a platform for the software development.

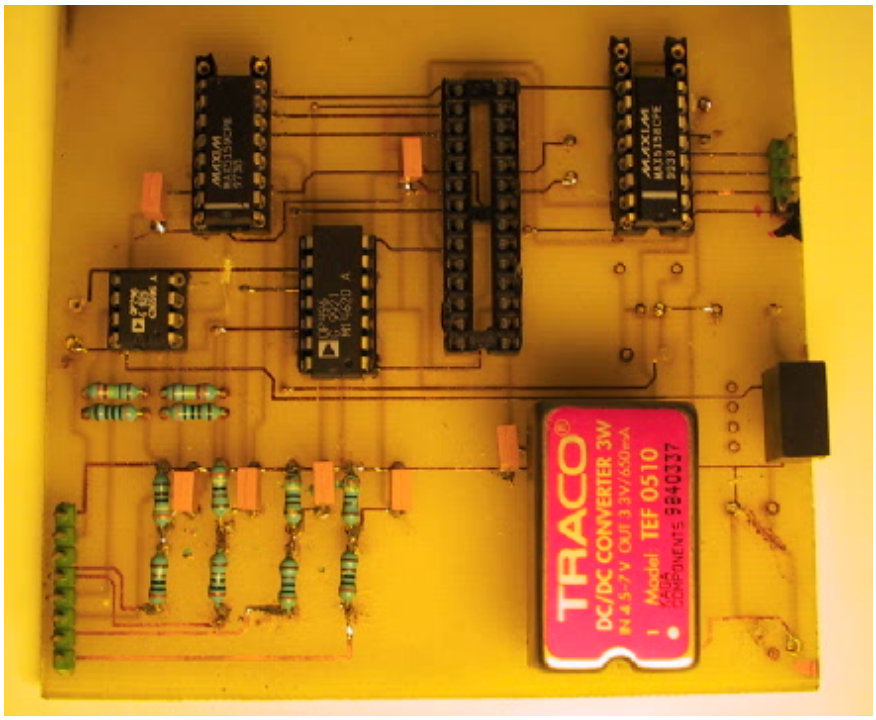


Figure 2.4: The first board that was constructed can be seen above. This board was made to study the problem and to develop the software early in the project. The picture is in natural size and the board has some drawbacks and design errors which are altered to the next board, which can be seen in Figure 2.5.

The second board is shown in Figure 2.5. It was quite hard to get all the components into this area, but the shorter the wires are, the less disturbances occur. The wires between the micro controller, the RS232 component, and the socket with the Bluetooth signals should maybe have been a little shorter to decrease the errors over the RS232 link. To connect the BCC to the Bluetooth module with sockets is though a smart choice, as the wires between the cards becomes as short as possible. The circuit board is shown in appendix B.

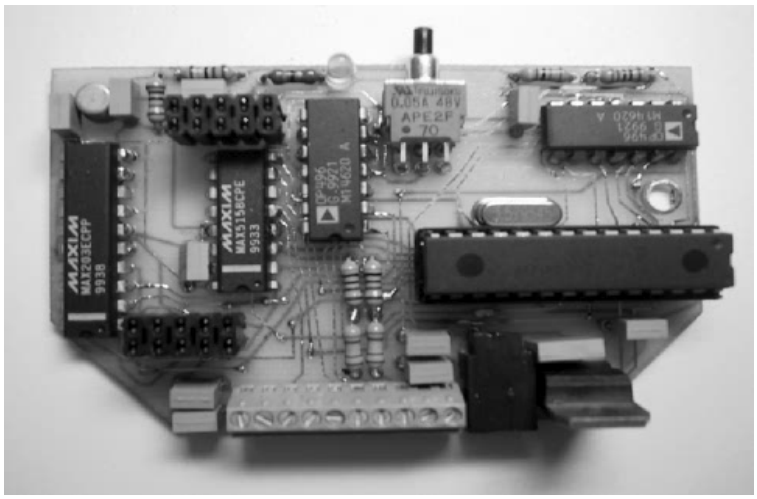


Figure 2.5: The second and final board is shown above in natural size. It has two sockets with ten pins each to left in the figure, which are used to connect the card to the Bluetooth module. The RS232 wires then become as short as possible. The small size of the board makes it less sensitive to disturbances. See appendix B to see the drawing of the circuit board..

2.2 Components

In this Section the different components are listed in Table 1 below. The total cost of the components on the final card is between 300 to 500 SKr.

Component	Elfa's stock no.	Quantity
Power Supply Socket, EIAJ RC 5320	42-053-57	1
Socket for Bluetooth Application Tool Kit	43-708-21	2
Socket for inputs and outputs	48-354-84	1
Micro controller, PIC16F876	Ordered from Memec	1
D/A-converter, MAX5158CPE	Ordered from Maxim	1
Operational amplifier, OP496	73-016-41	2
Voltage reference, ICL8069	73-101-21	1
Micro controller clock 3.5796 MHz, HC49/4H	74-515-03	1
Blue light diode	75-042-44	1
RS232, Max 203ECPP	73-229-77	1
Reset button, APE2F-6M-10	35-548-39	1
Voltage regulator, L7805CV	73-095-60	1
Capacitor, 1 uF	65-743-05	1
Capacitor, 100 nF	65-736-87	10
Capacitor, 4.7uF		1
Resistor, 5kΩ	60-730-76	2
Resistor, 5.4kΩ	60-731-18	2
Resistor, 6.8kΩ	60-732-33	1
Resistor, 10kΩ	60-734-23	1
Resistor, 470Ω	60-719-06	1
Resistor, 47kΩ	60-741-81	1
Resistor, 20Ω	60-703-20	1
Plastic distance	48-846-56	1

Table 1: The table lists all the components on the Bluetooth Control Card.

2.3 How to Access the Hardware

Figure 2.6 below shows where the pins, the reset button, and the power supply are situated on the BCC. It is easy to connect wires to the pins. A small screwdriver is needed to attach the wires, which then are safely connected to the card. The Bluetooth module can be pressed on the sockets that are shown to the left in the figure. The upper socket has the power supply and reset wire for the Bluetooth module and the lower socket has the RS232 signals. The socket for the power supply can be seen down to the right and the plug for that socket can be bought from a component retailer, for example Elfa. This plug usually sits on battery eliminators, so it is easy to supply the card with power. When the wires are attached and the card is supplied with power, the card waits for another Bluetooth device to communicate with it.

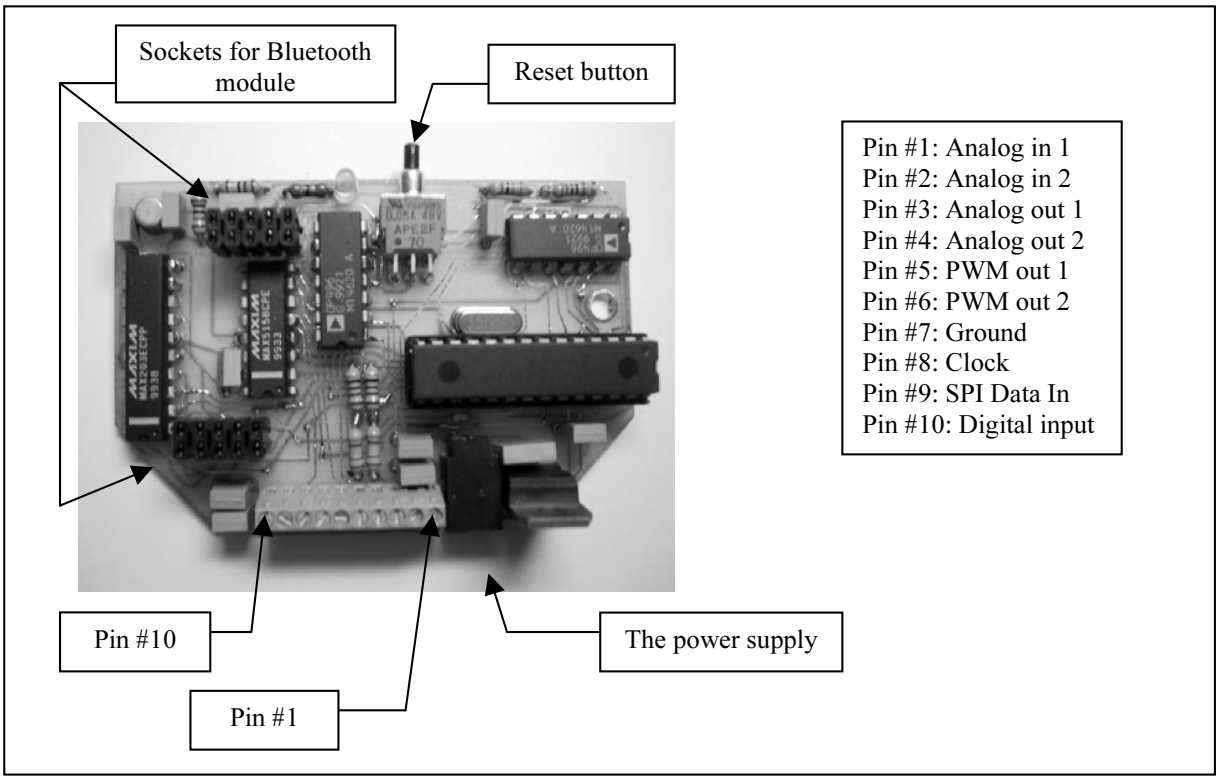


Figure 2.6: The Figure shows the BCC and how it is accessed. The two sockets to the left is connected to the Bluetooth module. The reset button sits on top of the card and when it is pressed the blue light diode to the left of it flashes. The 10 pins with the inputs and ouputs are situated on the bottom of the card and the figure shows where the different inputs and outputs are. The power supply is to the right of the pins and and the plug for it can be bought from a component retailer.

2.4 Development Environment

To develop the hardware of the BCC the CAD program Eagle was used. The card was then etched at the Department of Automatic Control at LTH and assembled at Sigma. The light edition of Eagle was used, because the BCC is small enough and has only two layers of wires.

The design of cards in Eagle contains three different stages: design of components, making connections between these, and placing them on the board. It is not necessary to design all the components on the card, since a lot of them already exist in Eagle, e.g. resistors, capacitors, and the standardized packages. Designing components is very easy, but it can however be a little tricky to redesign them, because components that are used can not be redesigned. That will say, if a component is placed in a view so connections to other components can be made, and it shows out that the component must be altered it is not permitted to have it in a view

where connections can be made. Therefore it has to be removed from all boards before it can be redesigned or a new component must be made.

After the components are designed they are placed in a “schematic view” and electrical connections are made between them. Eagle is very user friendly and to check that the connections are correct there is an “eye” that can be used. It is a special feature of the program that makes it possible to see which pins that are connected with each other. This is very powerful and it is recommended to use this final check before the last stage.

In the last stage the components are transferred to the “board view”. When the placement of the components is done, it is possible to either route the wires by hand or to let the program auto route the wires. In BCC’s case there are a lot of wires and the area is quite small so the auto routing could route 70 percent and the rest was routed by hand. The drawing of the wires on the circuit board was printed on a slide with a laser printer, see Appendix B, and then the card was etched. It has to be quite good quality of the printed slide in order to make a good etching.

3. Software Solution of Bluetooth Connection

The software is designed and implemented as a master to slave solution. The PC is the master of the connection and the BCC card is the slave. They are both refereed to as the host with the Bluetooth module as the host controller. The hosts refer to each other as remote devices. Figure 3.1 below shows the general design of the system.

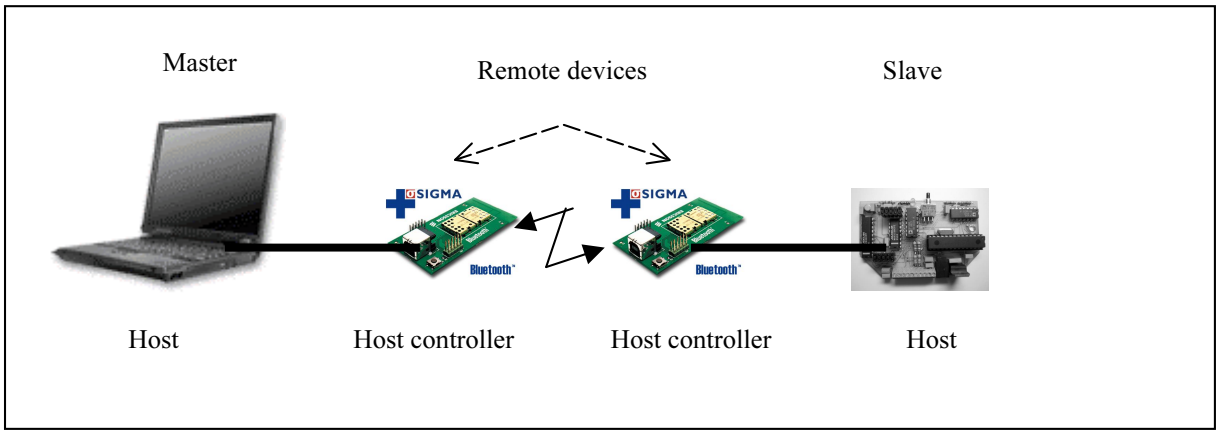


Figure 3.1: The general design of the system is a master and slave solution. The PC is the master and the Bluetooth Control Card (BCC) the slave. The master contains all the logic of the Bluetooth connection. The slave samples its inputs at a user defined sample period. The input values are then transmitted to the master over the Bluetooth channel. The packages that are sent from the master to the slave set the outputs of the BCC.

All the logic for the Bluetooth connection and control of the BCC card is gathered in the master. This makes the BCC card very generic and allows it to be attached to many different applications.

The master initialized the Bluetooth connection and data packages can be sent from the master to the slave and vice versa. This means that that the master can measure input signals and set different output signals at the slave.

The Bluetooth package type chosen for the Bluetooth connection is a DM1 package, see Appendix F Section 3. This package can transfer data payload between 0-17 bytes with full error coding at the symmetric max rate of 108.8 kbps over the air. A typical DM1 package is shown in Figure 3.2.

3.1 Software in Master

The software is developed in the program language Java. Java's main advantage is the machine independent structure of the language. It is also easier, compared with C++, which makes it more time effective to work with. The functionality is not constrained in any way using Java so it was the natural language to choose. Another advantage with Java is the communication package JAVACOM that simplifies communication with the serial port.

The software is designed as a Bluetooth stack with its different layers. The lower layers provide different services to the above layer. The benefit of dividing the program in different separated layers is an easier task of adding functionality. Different applications access the stack through an Application Programmable Interface (API). The API offers all functionality needed for hardware independent wireless communication. The ability to access the stack in a well-defined way makes it easy to develop new software, as well as modifying existing programs to gain the functionality of wireless communication.

All received packages are reassembled from the byte stream and then scheduled in a Java thread. The thread copies the data contained in the package from the general receiver buffer, into its own memory space. The data is maintained in a vector without any other copying needed during stack executing. This is done using the vector as a parameter together with a pointer to the first byte. Above layers simply peels off its own data by adjusting the pointer. Sending data is done in the same way. The top layer reserves all memory needed in a byte vector and copies its data into the vector. A pointer is used to keep track of the first byte and no extra memory allocation and vector copying is needed. This is illustrated in Figure 3.2.

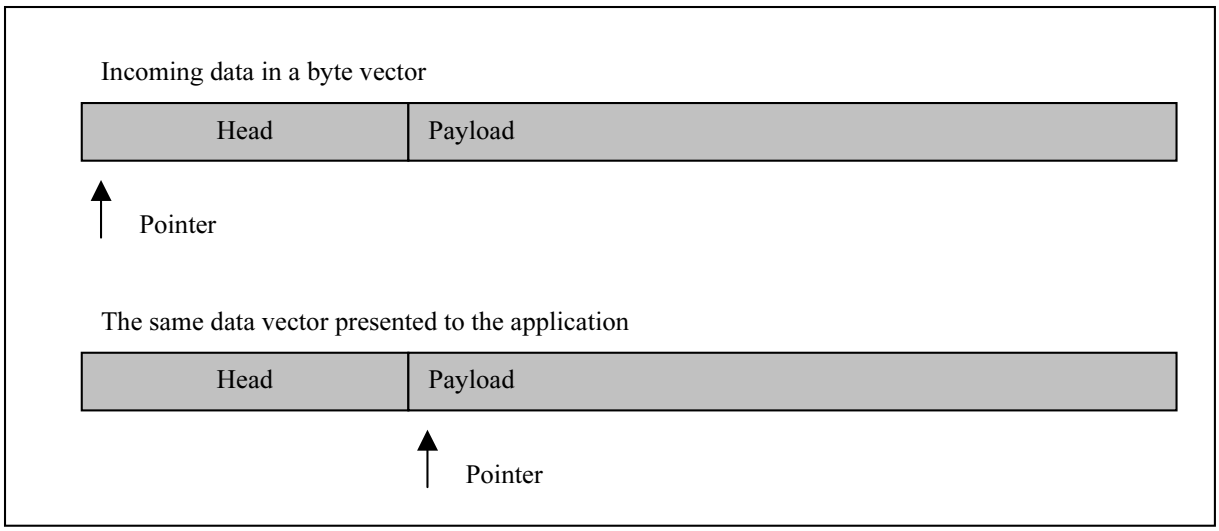


Figure 3.2: The Figure shows how incoming data packages are threaten by the master. The data of a package is copied to a data buffer in a software thread. The thread contains a pointer to the first correct byte. This pointer is changed during execution when different software layers "peel of" its data bytes. No extra data copying is then needed during execution.

The software in the master thesis is designed for communication with multiple remote devices on several channels. Due to the lack of support for multiple connections in Bluetooth firmware from Ericsson, the software is not implemented to support multiple devices. Multiple channels are not either supported, but are not a major task to implement on the current design.

The software stack is a prototype and has no claim of following the Bluetooth specification 1.0 B [2]. The main reason for this is the time constraint in the master thesis project.

3.1.1 Design

Figure 3.3 below shows the design of the Bluetooth stack. The design is done according to the Object Oriented Modeling and Design OOMD [1] process.

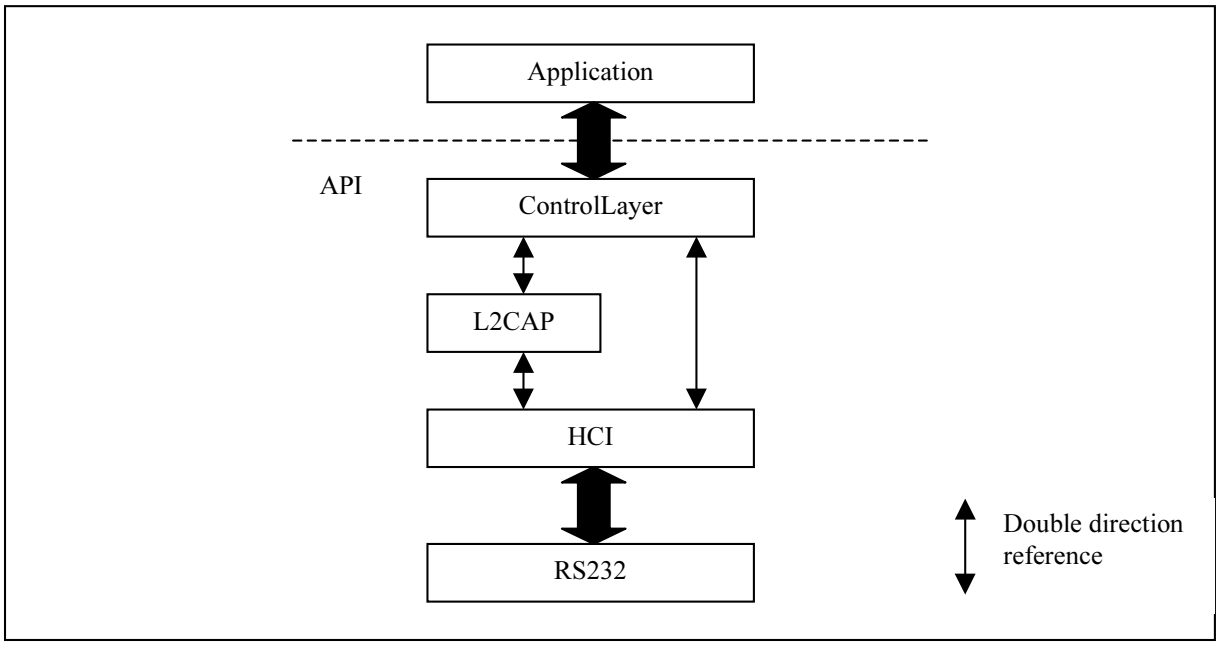


Figure 3.3: The Figure shows the design of the Bluetooth stack used in the master. Different layers access each other according to the arrows. Applications access the stack through an Application Programmable Interface (API). The API provides a user with well-defined methods and simplifies the development. The stack communicates with the Bluetooth module over the serial port RS232.

Description:

- Application
  A user developed application running on top of the stack gaining the increased functionality of wireless communication.

- Application Programmable Interface, API
  The API is a Java interface that is implemented by the application. It provides well-defined methods between the stack and the application. The benefit is rapid application development.

- ControlLayer
  The layer performs multitask functions, provided to the application by the API. An example is the initialization of the Bluetooth host controller that is performed in several steps. These methods are written with the goal of hiding the Bluetooth specific parts from the user.

- Logical Link Control and Adaptation Protocol, L2CAP
  The layer adds the functionality to send data packages over a Bluetooth channel. For more information on the L2CAP layer se appendix F Section 3.

- Host Controller Interface, HCI

The layer adds the general functionality of communication with the Bluetooth host controller. The Bluetooth host controller is accessed through the HCI layer. The layer is also dividing incoming packages in two categories. Data packages are sent to the L2CAP layer and event packages to the ControlLayer. For more information on the HCI layer see Appendix F Section 3.

- RS232
  The class establishes communication with a serial COM port. The byte stream that is received from the serial port is reassembled into data or event packages, and sent to the HCI layer. Data or command packages that are received from the HCI layer, are transmitted over the serial COM port.

3.1.2 Application Programmable Interface, API

The application implements the API. This means that the application has to implement the following methods, which are called upon by the stack.

- void receiveData(byte[] data,int pos)
  The method is called each time a data package is received. The integer variable pos points to the first data byte in the vector data.

- void receiveEvent(byte[] event)
  The method is called each time the stack receives an event from the host controller that it does not understand. The user should write the event to a debug window.

- void BTFound(byte[] adress)
  This method is called for each remote device found during inquiry. The vector contains the unique Bluetooth address of the remote device.

- void connectionIsClosed(byte[] btAdress)
  This method is called if the remote device closes the connection

- void stackInformation(String t)
  During stack execution this method is called with an explaining text. The method is essential for both debugging reasons and knowledge of the work being done in the stack. The text starts with "I ->" for information messages and "E ->" for error messages.

- void receiveRSSI(byte value)
  The user can call the stack and ask for the strength of the radio connection. The stack returns the difference between the measured Received Signal Strength Indication (RSSI) and the desired RSSI of the radio. Range –128 <= value <=128 with the unit dB.

The following methods are located in the ControlLayer and are the functionality gained from the stack.

- boolean init()
  The method is blocking and returns true if Bluetooth initialization was a success, otherwise false. If the Bluetooth host controller is not properly attached to the serial port, or if the host controller hardware is not in reset mode, the thread will never return. The method can be called multiple times during initialization, returning all threads in case of a success.

- boolean inquiry()
  The method is blocking and returns true if Bluetooth inquiry was a success otherwise false. For each found remote Bluetooth device the stack calls Btfound. The inquiry length is 6.4 seconds and the thread is always returned after that time period.

- byte[] createConnection(byte[] adress)
  The method is blocking and returns the channel identifier (CID). The CID is used to specify the unique data channel. The return vector is two bytes long and will, if a channel was established, have a value not equal to zero. The inparameter is the Bluetooth address of the remote device.
  A remote device cannot be connected if it is not found during an inquiry, and the inquiry must be completed before it tries to connect. After inquiry the remote device is stored in the stack and can then be connected without any further inquiries. If the Bluetooth controller has not received an answer within 5.12 seconds, it will timeout and the thread will be returned.

- void sendData(byte[] CID,byte[] data)
  The method sends data payload over the created channel. If any error occurs while transmitting the data, the method stackInformation will be called with an error massage.

- void closeConnection(byte[] btAdress)
  The method closes the connection with a specific remote device.

- void readRSSI(byte[] btAdress)
  The method measures the radio signal strength of the connection. The value is then compared with the ideal radio signal strength and returned with a call to the method receiveRSSI.

- reset()
  The method invokes software reset on the Bluetooth controller.

3.1.3 Implementation

The implementation beside from the API below is found in Appendix D.

```
interface API {

        // Bluetooth stack to application
        void receiveData(byte[] data,int pos);
        void receiveEvent(byte[] event);
        void BTFound(byte[] adress);
        void connectionIsClosed(byte[] btAdress);
        void stackInformation(String t);
        void receiveRSSI(byte value);

        /**
        *  Methods that the application can call in the Bluetooth stack
        * boolean init();                                   // blocking call
        * boolean inquiry();                                // blocking call
        * byte[] createConnection(byte[] adress);           // blocking call
        * sendData(byte[] CID,byte[] data);
        * closeConnection(byte[] btAdress);
```

```
        * readRSSI(byte[] btAdress);
        * reset();                                          // software reset of Bluetooth host controller
        */
}
```

3.1.4 Tests

All functionality provided by the API was tested and is working.

3.1.5 Future Improvements

A future improvement is to implement the stack according to the Generic Access Profile (GAP), which is specified by the Bluetooth specification 1.0 B [2]. When this is done the software can be tested and qualified as a Bluetooth product. The estimated time for this is around four weeks.

3.1.6 Development Environment

The software was developed on an ordinary PC using the Java virtual machine version 1.2.2 [12]. Access to the serial port was gained from the extension classes found in the serial communication package JAVACOM [12].

3.2 Software in Slave

The software was implemented in ANSI C and then downloaded into the flash memory of our micro controller. The program contains five different files that are compiled together. They have all well separated functions and provide each other with hardware independent services.

The software is very general and can control many types of processes. It can simultaneously be used for sampling inputs and setting outputs. For further information on the inputs and outputs see Section 2.3. The data measured at the inputs are sent over the Bluetooth communication channel in one package at a user defined sample period. The sample period is user defined between 6-255 milliseconds. All outputs are updated each time a valid data package is received. The protocol for transmitting and receiving data packages is described in chapter 8.

The general functionality of the software is illustrated in Figure 3.4. The micro controller is interrupt driven and sets different status flags depending on the interrupt type. The different interrupts can be individually shut off, but the status flags are still raised. This allows the developer to use both polling and interrupts.

The software in the slave uses different status flags to keep track of internal and external actions. An interrupt subroutine is called for every generated interrupt. The subroutine investigates the interrupt and sets different status flags depending on the origin. The main thread is polling a general event flag that is raised during an interrupt, such as the receiving of a data package. After the general event flag is raised, the main thread investigates other flags to establish the reason of the event, and take appropriately action.

Actions that are taken by the master thread are done by polling. An example is the transmission of data to the Bluetooth host controller, which is done byte by byte. A register is loaded with the byte and a special status flag is polled. When the flag is changed the register has sent the byte and is ready to transmit the next one.

An interrupt that occurs during execution in the interrupt subroutine is lost, so it is essential to keep the interrupt subroutine as small as possible.

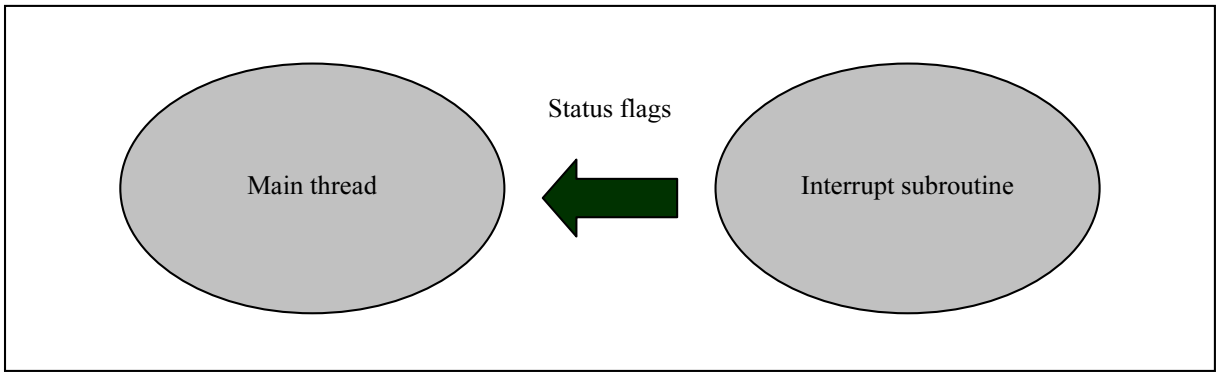


Figure 3.4: The Figure illustrates the general functionality of the software in the slave. The software samples its inputs at a user set sample period and updates its outputs. The inputs/outputs data is propagated over the Bluetooth channel. All the logic of the software is located in the main thread.
External events are detected by the interrupt subroutine, which upon detection of an event sets the correct status flag. The main thread is polling the status flags and takes the appropriate action when it sees a raised flag. The interrupt subroutine stores incoming byte of the Bluetooth channel in a general buffer and sets the new package status flag when it has received a whole package.

Human interaction with the software is provided by a blue flash diode on the BCC card. This diode flashes ten times after a hardware reset. If a not repairable software error occurs, the diode will start flashing, indicating the need of hardware reset. If the slave connects to another remote Bluetooth device, the diode will be turned on. Turning the diode off indicates the closing of the connection.

The software stack is just a prototype and has no claim of following the Bluetooth specification 1.0 B [2]. The main reason for this is the time constraint in the master thesis project.

3.2.1 Design

Figure 3.5 shows how the software is divided in different files.

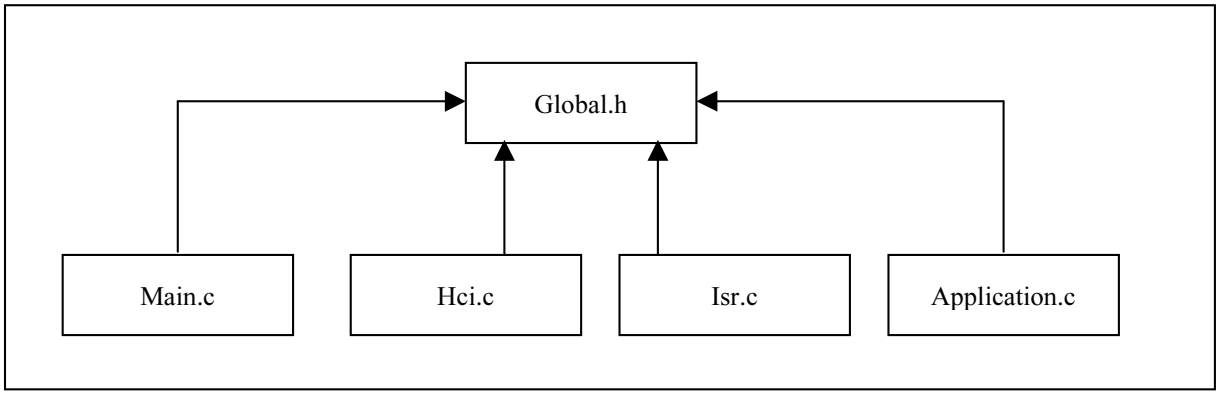


Figure 3.5: The Figure illustrates the structure of the software in the slave. The software solution is divided in five different files. The program starts in Main.c where the initialisation of the Bluetooth channel also is defined. The file Hci.c contains all Bluetooth specific information and provides the other files with Bluetooth independent services. Isr.c contains the interrupt subroutine and Application.c the logic of the software. The file Global.h defines all global variable and methods and is included in the other files.

File description

- Global.h
  The file contains a list of all global variables, functions and macro variables. The other files include this file.

- Main.c
  The program execution starts in this file. It contains the initialization code for the micro controller, Bluetooth controller and the application. The file Hci.c provides the initialization functionality of the Bluetooth controller. Main.c also creates the different status flags and most global variables used in the program.

- Hci.c
  The file contains all Bluetooth hardware dependencies. Initialization of the Bluetooth controller is done with different function calls. The calls are blocked and are not returned until the Bluetooth controller has answered with an event that indicates the success of the command. The file also provides the service of sending data over an L2CAP channel. A function call for sending a data package is blocked and the call is returned when the package is transmitted.

- Isr.c
  This file handles all micro controller interrupts. The file investigates the interrupts and sets different status flags. The decision on what action to take based on the interrupt is then left to the application. The interrupts that are used are received byte interrupt from Bluetooth controller on USART, three different timer interrupts, digital input interrupt and Serial Peripheral Interconnect (SPI) interrupt.

- Application.c
  The file contains all logic based upon different interrupts. The application investigates the status flags in a priority order, with the goal of minimizing the risk of overwriting old interrupt data.

3.2.2 Synchronization

All data to the different outputs are guaranteed to belong to the same data package. If incoming data corrupts an old not handled package, the old package is thrown away. This solution is essential if data structure larger than two bytes are used and guarantees atomic data.

The system clock in the micro controller is running at 3.5795 Mhz and the time to execute one instruction is one microsecond. The main mechanism for the synchronization is that all external events are much slower than the system clock. The interrupt routine is short and can very often execute the interrupt before the next happens. Multiple interrupt flags can be set.

There are two ways to lose an interrupt. The first is if the system does not have time to deal with an existing interrupt, before another with the same origin occurs. The other way is that two interrupts occur when already executing in the interrupt subroutine.

The system is deadlock free. Although the systems do not have time to take appropriate action on all incoming interrupts, the old interrupt and data are replaced with the most recently received. The execution of the interrupt routine is much faster then the occurrence of two interrupts with the same origin.

The received bytes from the Bluetooth host controller are stored internally in a two byte large FIFO queue. These bytes are copied to the receiver buffer in the interrupt routine. The interrupt routine sets a status flag when it has detected a whole package. This solution based on the incoming low speed of 57.6 kbit/s, guarantees that no bytes are missed.

A larger problem is the occurrence of corrupted bytes between packages, and the 2.9% large error probability, imposed by the micro controller, of the receiver channel (USART). The very high error probability depends on that the chosen speed is the highest supported by the micro controller. The corrupt bytes are easy to detect and remove, but the error probability can lead to the loss of synchronization of data and event packages. The largest package that the system expects is 20 bytes. Recognizing when too many bytes are received solves the synchronization problem. The system then resynchronizes by searching for the start byte in a data or event package. The system has no double buffering, due to the lack of memory, of incoming bytes and can loose Bluetooth packages when not synchronized. The loosing of synchronization has been tested and has shown to be of no real problem. A bigger and unsolved problem is the corruption of data on the USART channel that the high error probability imposes.

3.2.3 Implementation

The implementation is found in Appendix C.

3.2.4 Tests

All functionality is working and is tested with the exception of the SPI input. But the SPI input is implemented and has passed different code reviews. The reason for this is that the functionality of the application in the master thesis project changed during the development. The idea was to attach an ultra sound position sensor to the SPI input. The sensor was developed by others and could not be developed within the master thesis project's time plan.

3.2.5 Future Improvements

A future improvement is to implement the stack according to the Generic Access Profile (GAP), specified in the Bluetooth specification 1.0 B [2]. When this is done the software can be tested and qualified as a Bluetooth product. The estimated time for this is around five weeks.

3.2.6 Development Environment

The software is developed and debugged in the MAPLAB-ICD environment. MAPLAB is the main development environment for micro controllers [10]. The emulator ICD is a low cost emulator for the PICF87- family, which only costs 1500 SKr. The compiler used is HT-PIC [11], and this compiler is not included in MAPLAB-ICD. A range of diffrent compiles can be used with MAPLAB-ICD.

4. Communication protocol

The protocol for data packages allows a user to update all outputs or receive all input values in one package. The solution fits in one Bluetooth package, and thereby maximizes the data speed, and decreases the complexity of the software. It composes no constrain to the user functionality, other then unchanged outputs have to be written to new packages. The bytes are transmitted with the Least Significant Byte (LSB) first. Data fields bigger then one byte is also sent LSB first.

Figure 4.1 shows the general structure of a data package. The Payload is user defined and controls the BCC card. The head is added to the package in the stack and is of no interest to applications using the API.

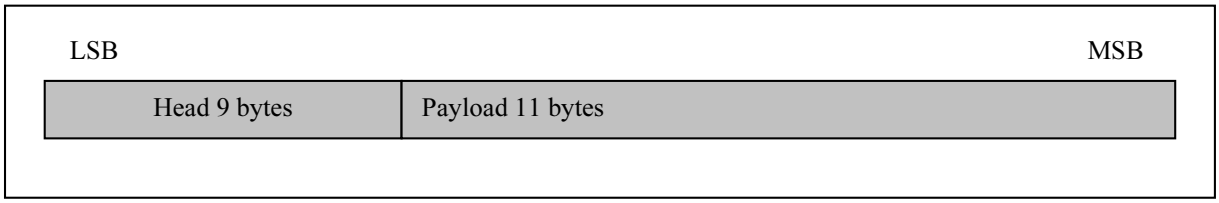


Figure 4.1: The Figure shows the general structure of a data package that is sent from master to slave or slave to master. The head is illustrated in Figure 4.2 and contains Bluetooth specific information in order to propagate on the Bluetooth channel. All data is contained in the payload field and can be seen in Figure 4.5 and 4.8.

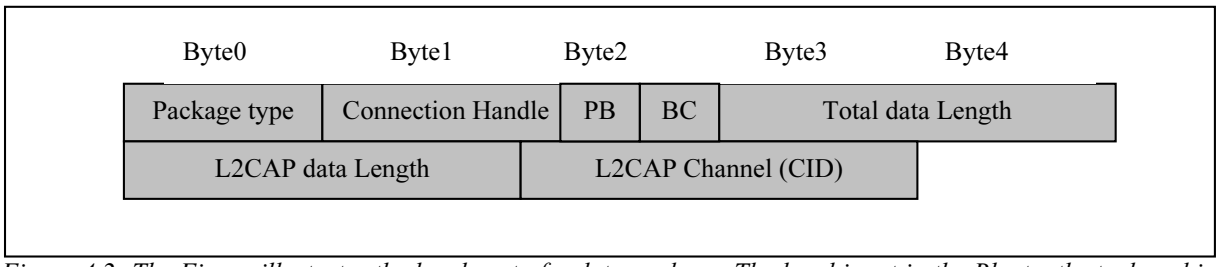


Figure 4.2: The Figure illustrates the head part of a data package. The head is set in the Bluetooth stack and is hidden to any application developer. The first five bytes contain information of the HCI layer and the last bytes the L2CAP layer of the Bluetooth stack.

The head is illustrated in Figure 4.2 and consists of the following byte fields:

- Package type
  There are four types of packages specified in the Bluetooth specification 1.0 B.

Value	Package type
0x01	Command package
0x02	Data package on a ACL connection
0x03	Voice package on a SCO connection
0x04	Event package

  The package type used here is the data package 0x02.

- Connection Handle
  The Bluetooth channel between two Bluetooth devices. The data field is 12 bits long.

- Packet Boundary flag (PB)

Value	Parameter description
00	Reserved for future use
01	Continuing Fragment Package of Higher Layer Message
10	First Package of Higher Layer Message (i.e. Start of an L2CAP package)
11	Reserved for future use

  The software does not support the splitting of data into multiple L2CAP packages. The field is 2 bits long and is always set to 0x10.

- Broadcast Flag (BC)

Value	Parameter description
00	No brodcast only point to point connection
01	Active Broadcast. Package is sent to all active slaves

10	Piconet Broadcast. Package is sent to slaves including slaves in “Park Mode”
11	Reserved for future use.

The Bluetooth firmware from Ericsson supports not either Piconets or broadcast. The field is 2 bits long and is always set to 0x00.

- Total data length
  The total data length specified in bytes including the L2CAP data length and channel fields.

- L2CAP data length
  The total length of the payload in bytes

- L2CAP channel (CID)
  The CID channel, of the connection.

4.1 Master to slave

The maximum number of packages per second that the master can send to the slave varies with the user set sample period at the slave, see Figure 4.3. This has not been tested in the master thesis project but is discussed in the text below.

At the highest speed the slave sends one package each 6 milliseconds to the master. Sending a package over RS232 is done at 57.6 kbit/s and takes 3.5 milliseconds, including the stop and start bits, leaving the processor 2.5 milliseconds when a sample period of 6 millisecond is used. During the 2.5 milliseconds, 2500 instruction cycles, the slave has to sample its inputs and update its outputs. Receiving and transmitting packages can occur simultaneously.

If the master to slave package rate is too high, the slave will not have time to update and sample its outputs/inputs and meanwhile receive and send bytes. The user set sample period will start varying, and received packages will be overwritten before they have updated the outputs.

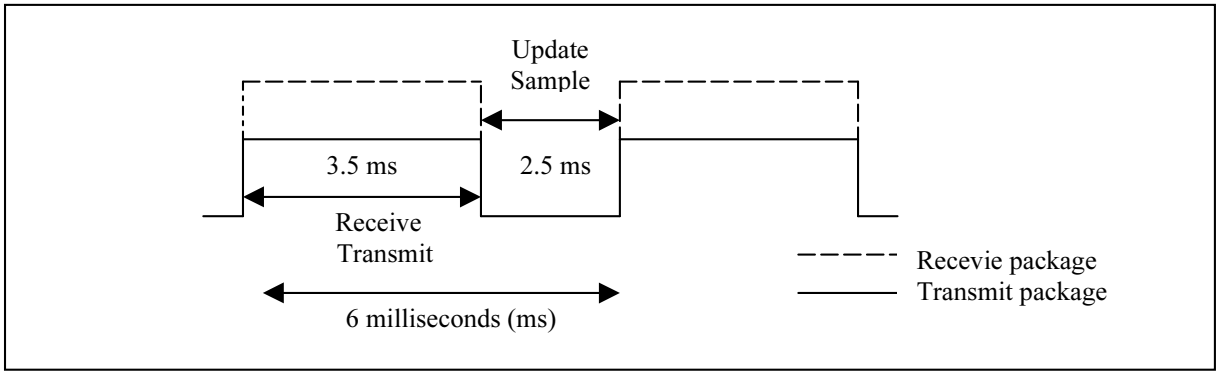


Figure 4.3: The Figure illustrates that the maximal dual speed of the system varies with the timing of transmitted and received packages in the slave. Packages are received and transmitted simultaneously at a maximum rate of 166 package/second. The time to update the outputs and sample the inputs before the next package is transmitted/received is at these rate 2.5 milliseconds. At the maximum data package rate the sample period can start varying depending on the timing. Received packages can also be overwritten before they have updated the outputs.

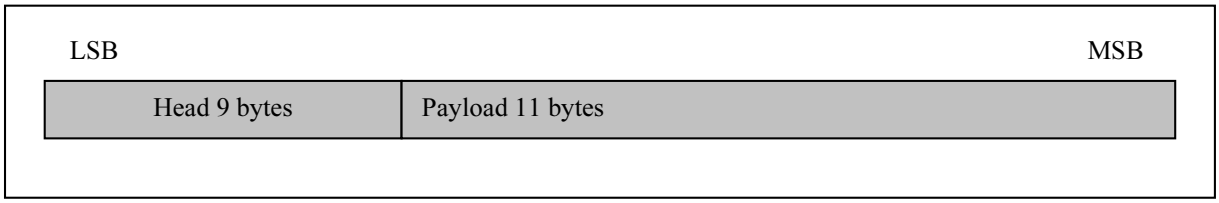


Figure 4.4: The Figure shows the structure of a master to slave package. The head is defined in Figure 4.2 and the payload in Figure 4.5.

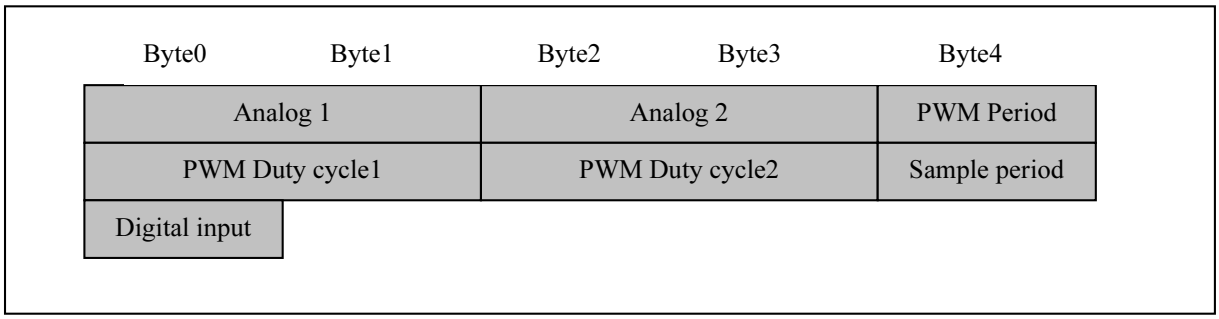


Figure 4.5: The Figure shows the data contained in the payload field of a master to slave data package, see Figure 4.4. Analog1 and Analog2 are analogue outputs at the slave. The PWM fields define two PWM outputs and the sample period byte is a user defined input sample rate that can vary between 6-255 milliseconds. The digital input byte enables or disables the digital input.

The payload is illustrated in Figure 4.5 and consists of the following byte fields:

- Analog1
  This field has 10 active bits. The bytes are sent LSB first and control the BCC's output pin 3. If all active bits are set to zero the output will be 0 V, and if all active bits set to one 5 V.

- Analog2
  This field has 10 active bits. The bytes are sent LSB first and control the BCC's output pin 4. If all active bits are set to zero the output will be 0 V, and if all active bits set to one 5 V.

The relation between PWM period and duty cycle is illustrated in Figure 4.6

- PWM Period
  The field sets the period length of the PWM outputs. The output range varies between 0-4 milliseconds in 255 discrete intervals. Pulse Width Modulation output is a feature of the micro controller, which means that the output values on BCC's pin 5 and 6 are exact and will not vary in time, no glitches. The period time of 4 milliseconds is achieved if the byte is set to 0xfe.

  The radio car application needs a period time between 18-25 milliseconds. This is solved with internal timers, which simulates the PWM outputs. Setting this byte to 0xff will give a period time of 22 milliseconds. The timers are interrupt driven and because of this the period time varies 256 microseconds.

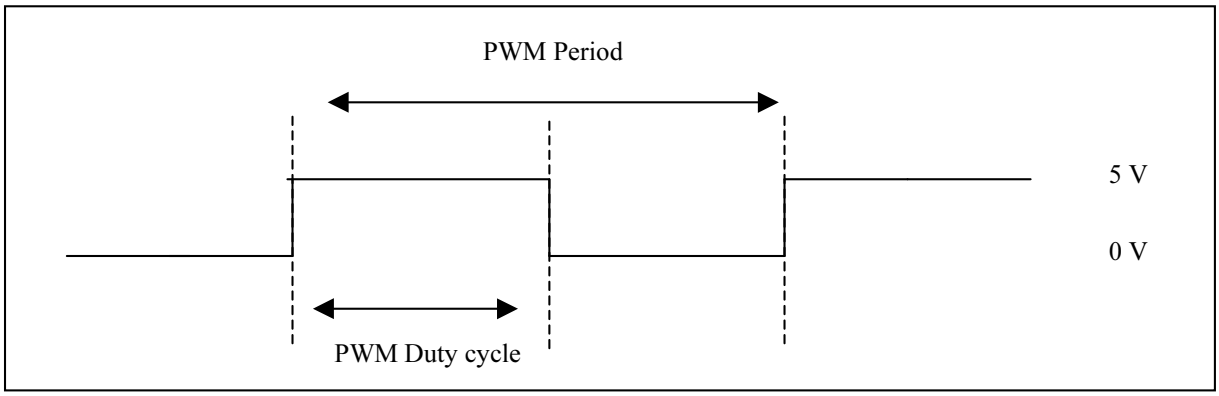


Figure 4.6: The Figure illustrates a general Pulse Width Modulating output. There are two PWM outputs on the Bluetooth Control Card (BCC). The output signal has a PWM period in which it can vary the voltage in two discrete intervals 0 or 5V. The PWM duty cycle specify the time of the signal when the output is 5 V, the remaining time of the period the output is 0 V. The PWM period is repeated and looks exactly the same if the user does not changed the duty cycle.

- PWM Duty cycle 1:
  The byte specifies the value of the duty cycle, which can vary between 0-4 millisecond. If the byte is set to 0xff the duty cycle will be 4 milliseconds.
  If the PWM period byte is 0xff, the duty cycles interval changes, and can vary between 0-2.38 milliseconds. This mean that if the PWM period byte is 0xff, and this byte 0xff, the duty cycle will be 2.38 milliseconds. The duty cycle can than not be chosen within the entire PWM period interval.

- PWM Duty cycle 2:
  The byte specifies the value of the duty cycle, which can vary between 0-4 millisecond. If the byte is set to 0xff the duty cycle will be 4 milliseconds.
  If the PWM period byte is 0xff, the duty cycles interval changes, and can vary between 0-4.56 milliseconds. This mean that if the PWM period byte is 0xff, and this byte 0xff, the duty cycle will be 4.56 milliseconds. The duty cycle can than not be chosen within the entire PWM period interval.

- Sample period
  The user can set the sample period of the inputs at the BCC card. The allowed sample period varies between 6-255 millisecond. If the sample period is less than 6 milliseconds, the software in the slave will change the time to 6 millisecond. The byte value 0x09 will set the sample period to 9 milliseconds.

- Digital input on
  If this byte is set to 0x01 the digital input will be enabled otherwise disabled. This allows the user to leave the digital input pin 10 unconnected. The SPI pin 9 can also be unconnected if the digital input is turned off. The clock output at pin 8 will always be on regardless of the value of this byte.

4.2 Slave to Master

The slave transmits packages to the master at a user defined sample period. The sample period can vary between 6-255 milliseconds, which gives a maximal rate of 166 packages/second. The packages contain all input values of the BCC card.

.

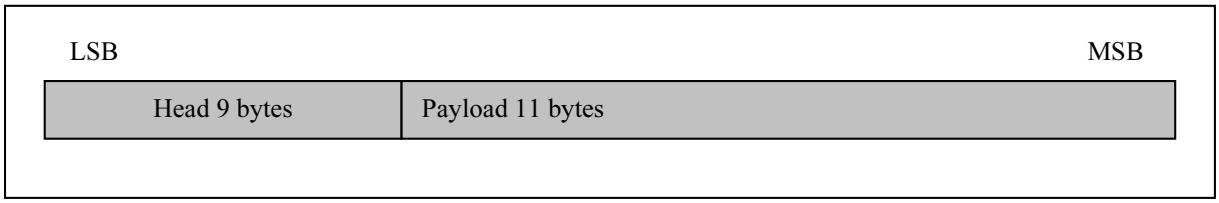


Figure 4.7: The Figure shows the structure of a slave to master package. The head is defined in Figure 4.2 and the payload in Figure 4.8.

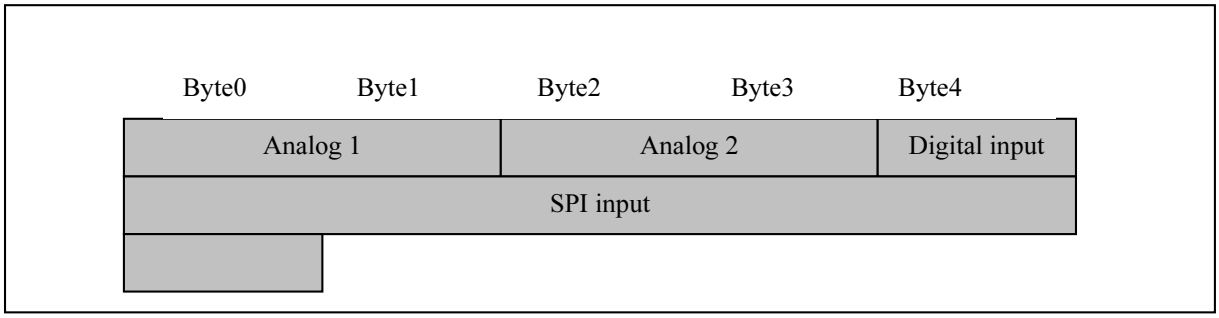


Figure 4.8: The Figure shows the byte contained in the payload field of a slave to master data package, see Figure 4.7. Analog1 and Analog2 are analogue inputs. These inputs have 10 active bits that are sent Lest Significant Byte (LSB) first. The digital input byte is the digital interrupt and SPI input the digital input bytes. All inputs are sampled at a user defined sample rate that can be between 6-255 milliseconds.

Figure 4.5: The Figure shows the data contained in the payload field of a master to slave data package, see Figure 4.4. Analog1 and Analog2 are analogue outputs at the slave. The PWM fields define two PWM outputs and the sample period byte is a user defined sample rate that can vary between 6-255 milliseconds. The digital input byte enables or disables the digital input.

The payload is illustrated in Figure 4.8 and consists of the following byte fields:

- Analog1
  This field has 10 active bits, the bytes are sent LSB first and specify the input voltage at pin 1 on the BCC card. If all active bits are set to zero the input is 0 V, and if all active bits are set to one 5 V.

- Analog2
  This field has 10 active bits, the bytes are sent LSB first and specify the input voltage at pin 2 on the BCC card. If all active bits are set to zero the input is 0 V, and if all active bits are set to one 5 V.

- Digital interrupt
  The digital interrupt pin 10 is triggered on the raising edge. The detection of the raising edge will set this byte to 0x01 for one package. The next packages until a new detection will always be 0x00. If two raising edges occur before the master thread has taken action

on the first interrupt. The software will remove the first interrupt, and not send 0x01 in the next package.

- Serial Peripheral Interface (SPI) input
  The field is 6 byte large and contains the serial data received from input pin 9. The input pin 9 is enabled by a raising edge on the digital interrupt pin 10, and disabled on the next raising edge. The data is not written to the transmit buffer before the slave has received all 6 bytes, and is transmitted in the next data package. Information can be lost if the SPI input speed is greater than the input sample period for the slave.

5. The Bluetooth Controlled Beetle

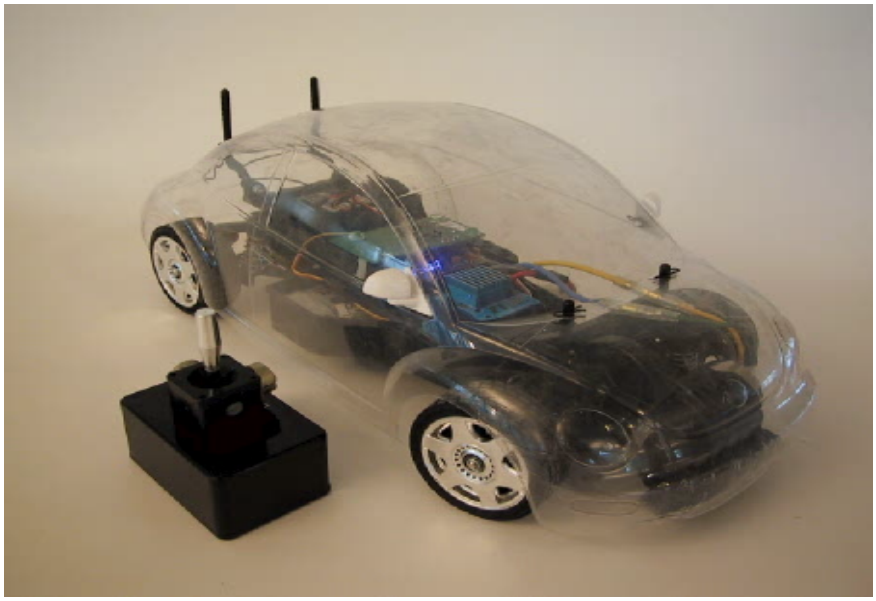


Figure 5.1: The Bluetooth Controlled Beetle is an application that is done to show the possibilities with the Bluetooth Control Card (BCC). The car is in 1/10th scale and is steered by the joystick next to it.

To show the possibilities of the Bluetooth Control Card (BCC) a car application was designed and implemented. It is a radio control car in 1/10th scale. The brand of the car is Volkswagen Beetle. It can be seen in Figure 5.1 together with the joystick that is controlling it. This application was chosen since Sigma wanted to have a fun Bluetooth demonstration to present on fairs and other events. A car is nice to show as the impression is visual and it is easy to convince people that the wireless technology works. Figure 5.2 shows how the control of the car is configured. The signals from the joystick are transmitted to the PC and then steer signals in the PC are computed and transmitted to the car. The car could of course have been steered directly from the joystick, but then the software in one of the BCC would have to be changed. In the PC it is also easy to add features such as limited speed or special control strategies.

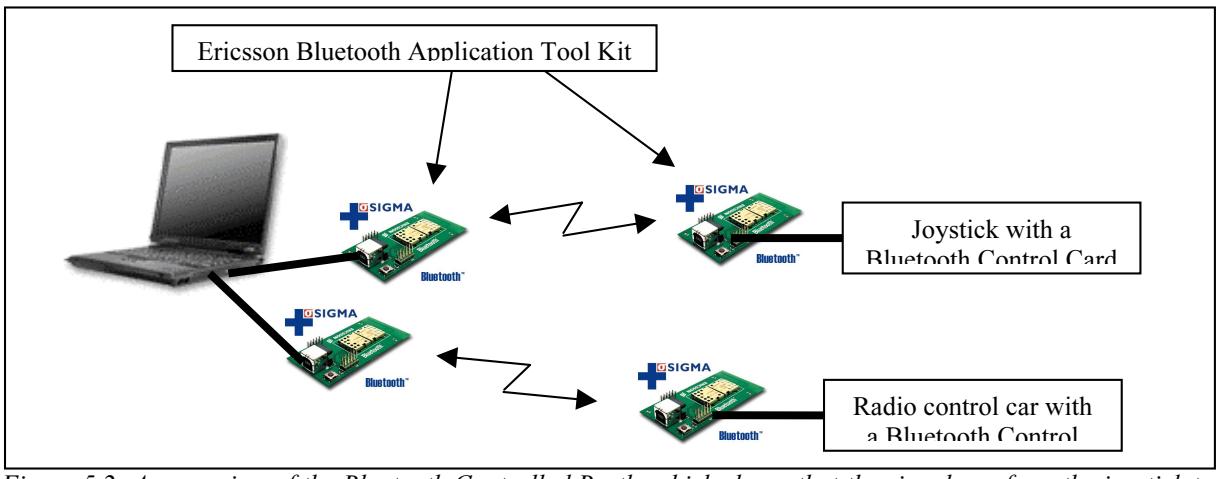


Figure 5.2: An overview of the Bluetooth Controlled Beetle which shows that the signals go from the joystick to the PC and then to the car. The car could of course have been steered directly by the joystick, but then the software in the BCC would have to be altered. In the PC there is an interface where the Bluetooth links can be controlled and it is also possible to limit the speed of the car.

5.1 Software Solution of Car Application

The software is the same as described in Section 3. The two Bluetooth modules at the PC are the masters. Two Bluetooth modules are needed at the PC, because the modules only support point-to-point connections so far. To communicate with the Bluetooth modules the API that was described in Section 3.1.2 is used. The API is quite easy to use and that speeds up the development of the application. The software is written in Java, which is a natural choice as the stack of the master is written in the same language. Java is platform independent and that is of course an advantage.

5.1.1 Design

The software in the PC handles the Bluetooth connections and the control of the car. The class diagram can be seen in Figure 5.3.

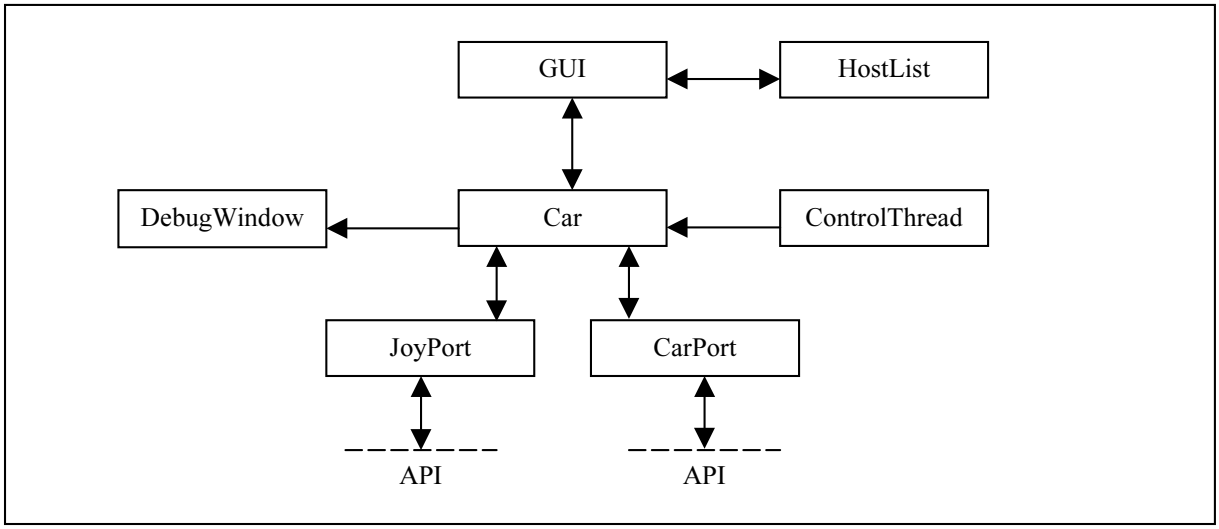


Figure 5.3: The class diagram of the software for the car application shows which classes that have a refence to the other classes. For example does ControlThread have a reference to the class Car and therefore the arrow goes from the class ControlThread to the class Car. The logic for the Bluetooth connections and the car control is handled by Car, which communicates through CarPort and JoyPort and displays the status in the Graphical User Interface (GUI). The GUI can be seen in Figure 5.4.

The stack's APIs communicate with CarPort and JoyPort. The class Car handles the logic of the Bluetooth connections and the car control. The program is started from this class and it creates the other objects. The debug information that comes from the stacks or the other classes is sent to the DebugWindow. It is possible to show all information or just the errors. It is very convenient to use the DebugWindow if something goes wrong, for example with the initialization of the Bluetooth modules or the establishing of the Bluetooth connections. The communication with the user is done through the Graphical User Interface (GUI), which can be seen in Figure 5.4.

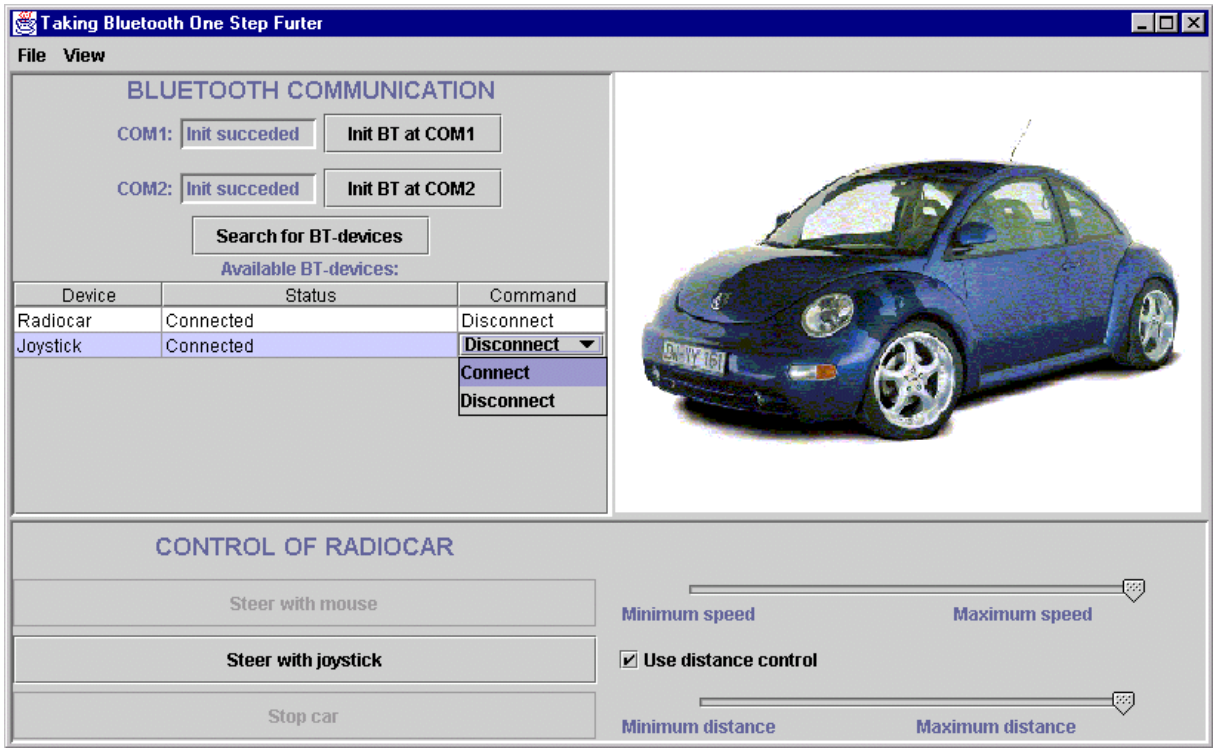


Figure 5.4: The Graphical User Interface for controlling the Bluetooth connections and the control of the car. The user begins to initialize the COM ports and then he or she searches for other Bluetooth devices. When the car and joystick are connected the user can choose to steer the car with the joystick. It is also possible to limit the speed of the car.

The user begins to initialize the Bluetooth modules and then he or she searches for other Bluetooth devices. Those units that answered are stored in the HostList and displayed by the GUI. It is then possible to connect to the car and the joystick. The car can then be steered by the joystick. When the class JoyPort receives information from the joystick it calculates the speed and steering and sends them to the class Car. The class ControlThread wakes up ten times per second and transmits the speed and steering to the car through the class CarPort.

5.1.2 Control Strategy

The PC has a very powerful processor and can be used for a lot of computing. Different control strategies can be implemented, for example could the PC automatically control the car if the car's position would be sent to it. In the master thesis project the car is only steered by a joystick and no automatic control is implemented. The application was designed so the car could be steered by the mouse, but this was never accomplished. However, the implemented control strategies are described below.

The BCC supplies the joystick's two vertical axes with power. The outputs are set to 5 V and steering and speed information is read on the analogue input pins as a voltage between 0-5 V. The steering and speed information is sent to the PC, which converts them to PWM signals that fit the car. These are then sent to the car. It is not sure that the packet that sets the ouputs to 5V reaches the joystick. Sometimes the voltages are set to zero, which is due to the bad RS232 link. To guarantee that the right voltages arrive, the master has to retransmit the package until it sees that the voltage received from the slave is not equal to zero.

The PWM modules in the slave could not have as long period as the car needed, so the PWM was remade in software. The remade PWM is interrupt driven, which makes the duty cycle vary randomly. This means that the control of the car becomes a little shaky. To make the car

completely still when the joystick is in its center position, the PWM are turned off and the signals from the joystick become flat.

If the joystick or car either is brought out of range or reset the connection is closed. The master is then automatically trying to reconnect. The closing of the connection is shown as a message in the status field in the GUI. The reconnect time can vary between 1-40 seconds and can often be improved by a software reset when the host is brought back in range of the master. The variation in the reconnect time is due to hardware constraints in the Bluetooth module that was available in the master thesis project.

The slave listens every 1.28 seconds for a connection attempt during 10.25 milliseconds. This is done on a special radio frequency, and that frequency is changed at the next attempt. If the master at the same time tries to connect, it has time to scan 16 of the 79 frequencies. If the master fails to connect it changes the scan frequencies. This means that there is a high probability that they miss each other. The search time of the slave can be adjusted according the Bluetooth specification 1.0 B, but this is not implemented in the Bluetooth module.

The application was also designed to have some kind of distance control. This was not implemented either. However, the distance to the car could perhaps be measured by checking the strength of the radio waves from the car. In the Bluetooth standard there is something called the RSSI value, which is proportional to the strength of the radio waves. When the RSSI value becomes low some kind of control action can be taken, for example can the speed be constrained or the car can be steered back. The RSSI functionality is provided by the API but not used in this application.

5.1.3 Implementation

See Appendix E.

6. PC Application controlled by BCC

Figure 6.1 shows the joystick that is used for controlling the application at the PC.

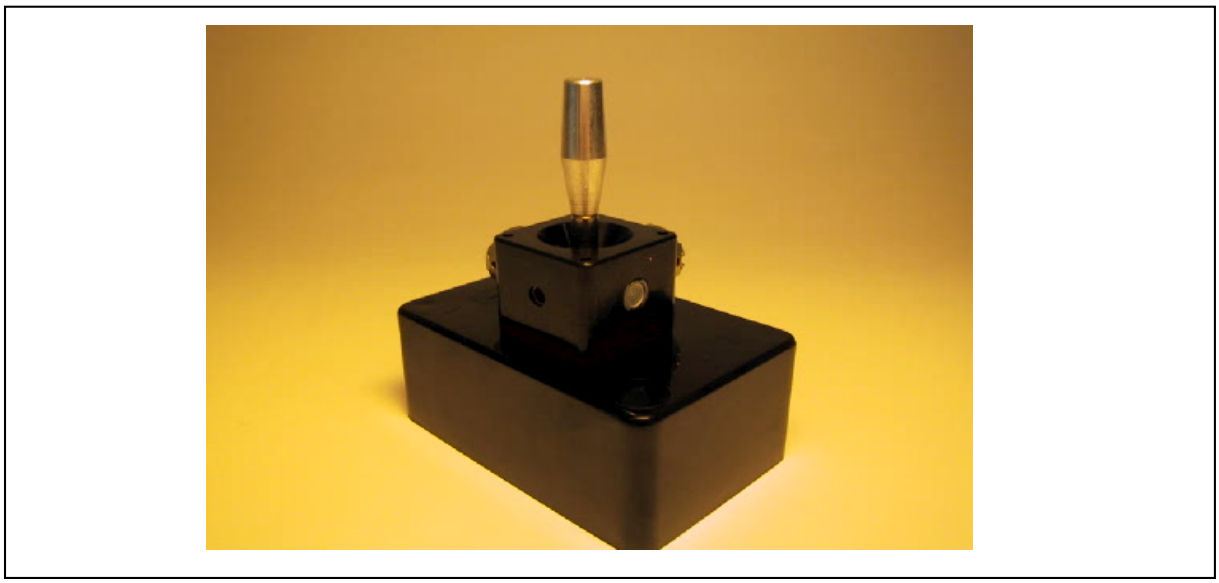


*Figure 6.1: The Figure shows the joystick that controls the application running on the PC. The black box is 8*4 cm large and contains the Bluetooth module, Bluetooth Control Card (BCC) and battery for power supply. When the joystick is steered the analogue values are sampled at the BCC and sent to the PC over the Bluetooth channel. The analogue values are then transformed in the PC to steer information controlling the application. A general picture of the system can be seen in Figure 1.1. Figure 6.2 shows a screen dump of the application.*

Figure 6.2 illustrates a screenshot of the application running on a PC. The software is written in Java and can without modification be executed on other hardware platforms.

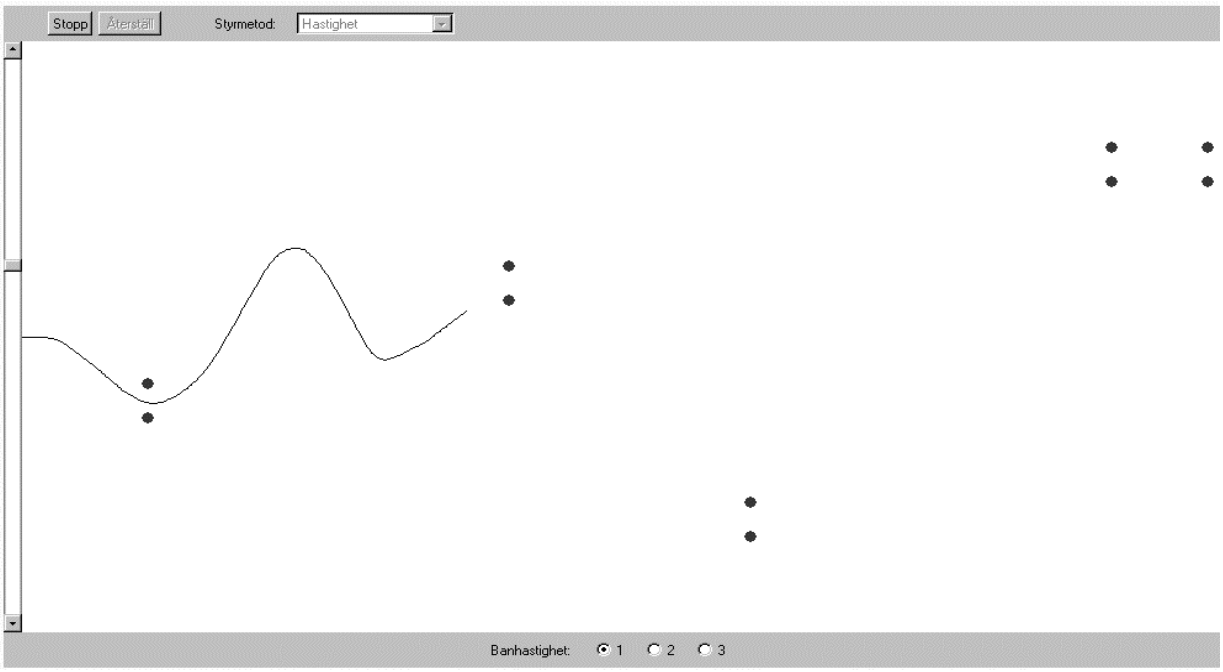


Figure 6.2: The Figure shows a screen dump of the application running on the PC. The line running from left to right shall be steered between the dots by the joystick in Figure 6.1. The line simulates different processes and by applying different regulators to the system the line will become easier to steer. The joystick can be passed around the audience up to a distance of 20 m. By visualising the benefits of different regulators automatic control can be made more popular, and the motivation of students studying automatic control courses can increase.

The application is developed at the Department of Automatic Control at Lund University, and modified to gain the functionality of wireless communication during the master thesis project. It shows how different types of regulators make it easier to control a process. The idea is to increase the motivation of students who are studying automatic control. The benefit of using a regulator is made clear by the application.

A joystick can control the application up to a distance of 15-20 meters. This allows the joystick to be passed around in the audience.

The application is written in Java and gains the wireless functionality using the API in the Bluetooth stack, se Section 3.1.2. The program is started with the serial port e.g. COM1 used as an in parameter. A Bluetooth module in reset mode should be attached to the port. The first thing that happens is that the Bluetooth stack tries to initialise the module. After the initialisation has succeeded it starts an inquiry after remote Bluetooth devices and connects to the first found.

In an environment with multiple Bluetooth hosts, it is possible that it connects to another host then the one in the joystick. This has to be solved by implementing the specific Bluetooth address of the host in the joystick and only allow connections to that address. This feature has not yet been implemented and is left to the user of the application. Debug and stack information is written to the Dos window.

The joystick is attached to the BCC. The card supplies the joystick's two vertical axes with power. One output is set to 5 V and steering information is read on the analogue input pin as a voltage between 0-5 V.

The input is sampled each 50 milliseconds and transferred to the master. In order to supply the joystick with power, a correct package has to be received at the joystick. To guarantee this, the PC has to retransmit the package until it sees the voltage change in the next package, received from the joystick.

Human interaction with the hardware and software is provided by a blue flash diode on the BCC card. This diode flashes ten times after a hardware reset. If a not repairable software error occurs, the diode will start flashing, indicating the need of hardware reset. If the joystick connects to the application, the diode will be turned on. Turning the diode off indicates the closing of the connection.

If the connection is lost, because the slave is out of range or a reset, the master will start trying to reconnect. This issue is discussed in Section 5.1.2. If the slave is brought back in range of the master the reconnect time can vary between 1-40 seconds. This time can often be improved if a reset is done to the slave.

7. Conclusions

In the master thesis project a generic I/O unit has been constructed. This platform can be attached to sensor and actuators in industrial processes, gaining the functionality of wireless communication provided by Bluetooth. To demonstrate the ability of Bluetooth and the generic I/O unit, a radio car application has been developed.

The application has gained a lot of interest. It will be presented at the Bluetooth Developers Conference in San José USA [4], with 1500 persons in the audience. When the master thesis project is finished, it will be used at the Department of Automatic Control at Lund University to control different processes. Sigma Wireless will also use the car application to demonstrate the Bluetooth technology and to market the company worldwide. The master thesis project has been completed within the planned 20 weeks.

Bluetooth is a well-suited technology for automatic control. Problems like retransmission and lost connection can easily be dealt with. But the communication speed is a constraint compared with other technologies, see Appendix F Section 5, which makes Bluetooth less suitable for fast and time critical systems.

Using the existing Bluetooth Application Tool Kit [3], the development can be done simultaneously in both hardware and software. The achievement of the master thesis project shows that Bluetooth is a mature technology.

We think the future success of Bluetooth depends on when the price of 5 US dollars per Bluetooth module can be achieved. This will make the Bluetooth concept cheaper than cables in an office environment. Bluetooth will then automatically be included in e.g. mobile telephones and computers.

Although there are a lot of competitors to Bluetooth, which have better performance in single areas, there is no one that already is supported by 2000 companies, nor with the aim of becoming a worldwide wireless standard supporting both data and voice.

8. References

[1] James Rumbaugh, ” Object Oriented Modeling and Design”, 1991, Prentice Hall.
[2] www.bluetooth.com/developers/specification/specification.asp
[3] www.comtec.sigma.se
[4] bluetooth.jli.net
[5] Göte Andersson, “Bredbands konkurrens i sikte för Bluetooth”, Elektroniktidningen, 20 Okt, 2000
[6] Dagmar Zitkova, “Genombrott för radio-lan som klarar 11 Mbit/s”, Elektroniktidningen, 20 Okt, 2000
[7] Eve Ekelöf, “Fel i specifikationen försenar Bluetooth”, Elektroniktidningen, 20 Okt, 2000
[8] Lisa Ringström, “Svårt och dyrt att få fram fungerande blåtandsprylar”, Elektroniktidningen, 20 Okt, 2000
[9] Magnus Ewert, “Datakommunikation Nu och i framtiden”,1998, Studentlitteratur
[10] www.microchip.com
[11] www.htsoft.com
[12] java.sun.com/products

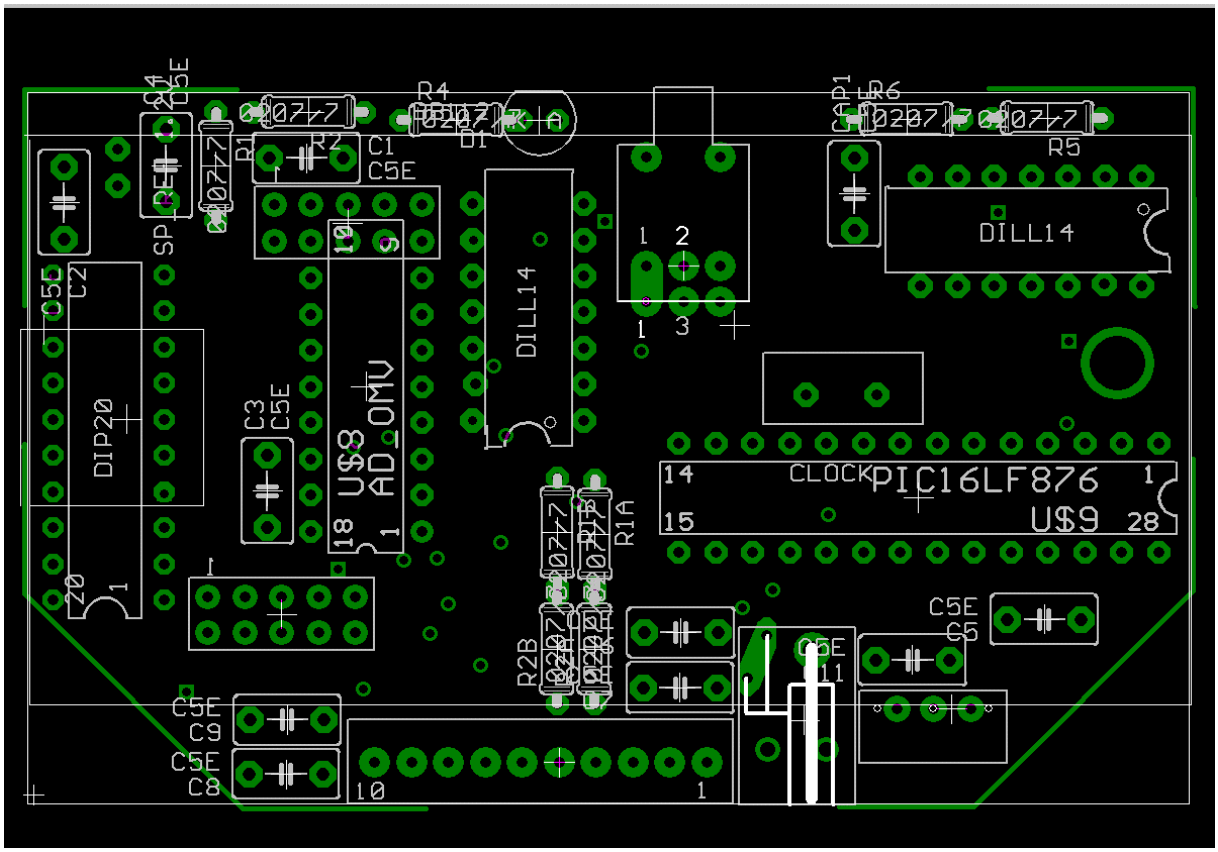
General references:
www.bluetooth .com

Appendix A: Time Planning

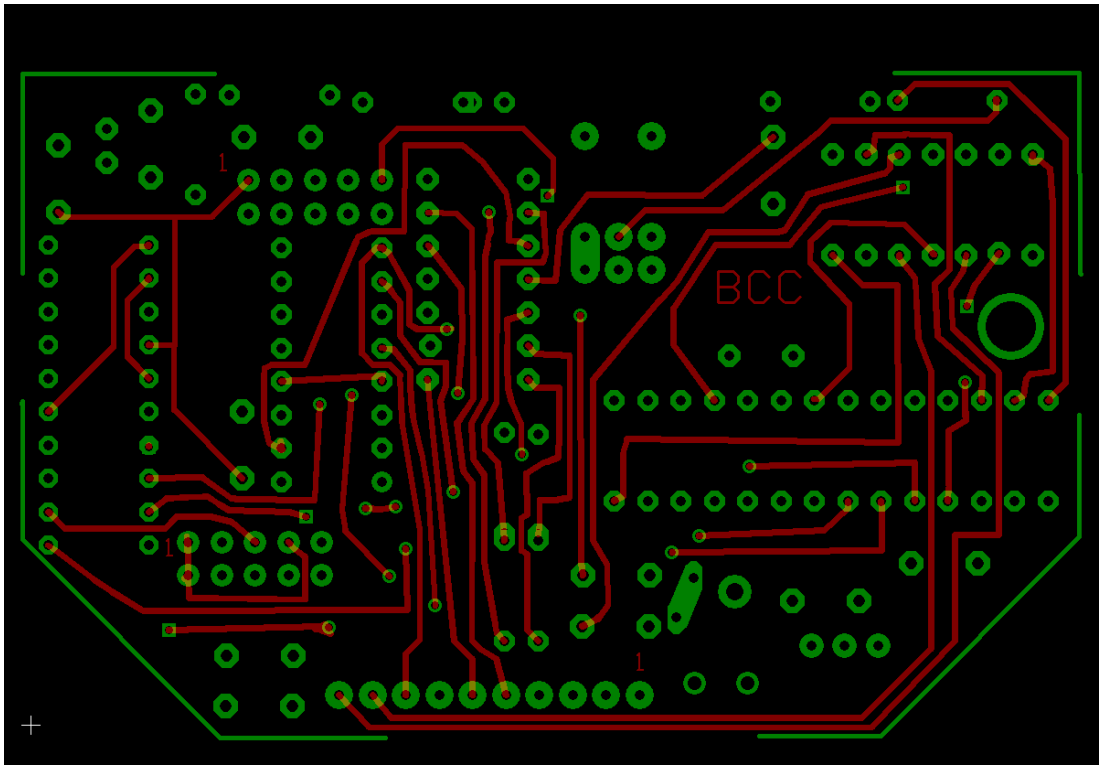
Time Planning	Commitment	
		P=Part time week, 3-4 days F= Full time week
Week		**Activity**
16	P	Study Bluetooth specific information
17	P	Study Bluetooth specific information
18	Vacation	
19	School	
20	School	
21	School	
22	F	Design of prototype
23	F	Design/Cad of prototype
24	F	The construction of the prototype
25	F	General software development
26	F	General software development
27	F	Communication between prototype and Bluetooth
28	F	Design of BCC
29	Vacation	
30	Vacation	
31	F	CAD/construction of BCC
32	F	Software development in slave
33	F	Software development in slave
34	P	Software development in master
35	P	Software development in master
36	P	The development of PC controlled application
37	P	The development of car application
38	P	The development of car application
39	P	The development of car application
40	F	Master thesis report
41	F	Master thesis report
42	F	Reserved time
43	F	Presentation and demonstration of the master thesis project on ARKAD 7-9 november

Appendix B: Circuit Board

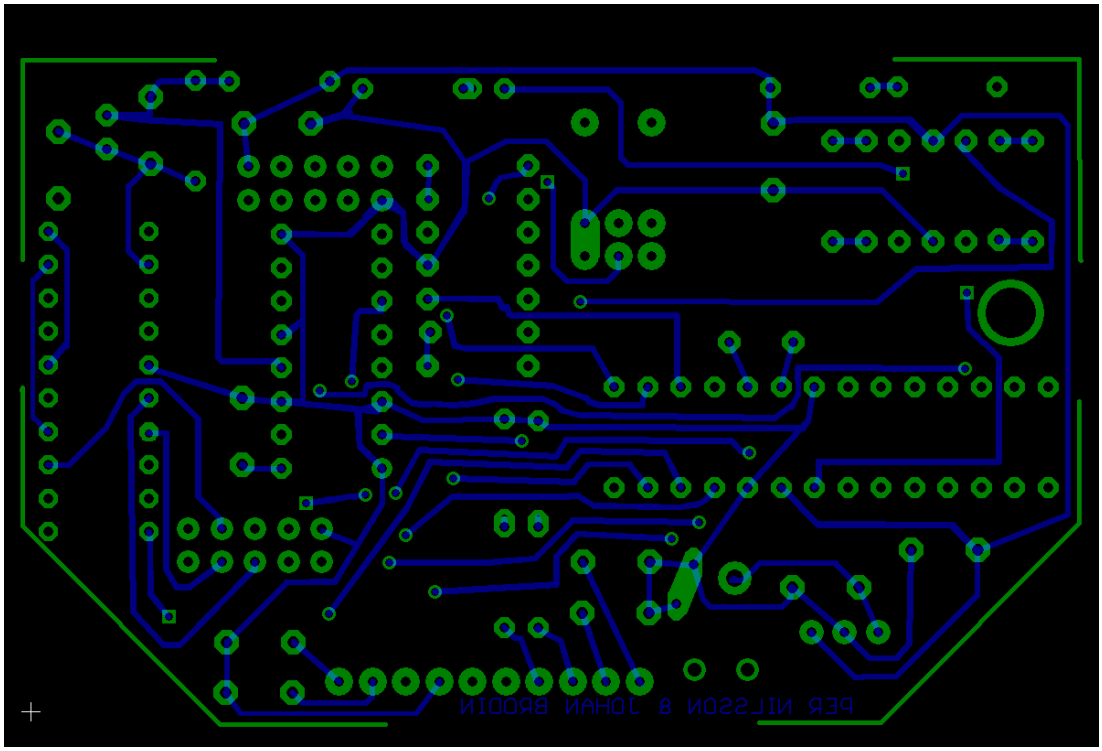
The components' location on the circuit board is shown below. The size is 8 * 5 cm.



Top layer, the size is 8 * 5 cm.



Bottom layer, the size is 8 * 5 cm.



Appendix C: Code in slave

```
---------------------------------------- Global.h ----------------------------------------
#ifndef GLOBAL_H
#define GLOBAL_H

typedef unsigned char BYTE;
typedef union
{
   struct S1                              // control bits, 0 disable, 1 enable
   {
      unsigned int global_interrupt:1;    // general interrupt bit
      unsigned int transmit_allowed:1;    // transmitting on UART is allowed
      unsigned int timer_interrupt:1;     // timer0 interrupt
      unsigned int receive_interrupt:1;   // receive interrupt on UART
      unsigned int spi_interrupt:1;       // spi interrupt
      unsigned int spi_sending:1;         // sending on on spi
      unsigned int digital_interrupt:1;   // interrupt on digital input
      unsigned int init:1;                // the system in init bluetooth
      unsigned int connected:1;           // bluetooth is connected
      unsigned int new_data:1;            // new data is read. For synchronization
   }bit_name;
}system_reg_t;


/************* Main.c************************************/
extern bank1 system_reg_t system_reg;       // control bits
extern bank1 unsigned int buffer_num;       // maximal number of package in buffer
extern bank1 unsigned int package_size;     // maximal number of byte in package
extern bank1 BYTE receive_buffer_index;     // possition in receive vector
extern bank1 BYTE transmit_buffer_index;    // position in receive vector
extern bank3 BYTE receive_buffer[80];       // Define receive_vector_1
extern bank2 BYTE transmit_buffer[20];      // Define transmit_vector
extern bank2 unsigned int package_length;   // length of receiving package
extern bank2 unsigned int transmit_length;  // length of transmitting package
extern bank2 BYTE connection_handle[2];     // connection_handle
extern bank2 BYTE digital_value;            // value of digital input
extern bank2 BYTE SPI_value;                // value of SPI input
extern bank2 BYTE CTS;                      // flow control  from RS-232

extern void wait(unsigned int count);

/***************** isr.c ************************************/
extern bank1 BYTE car_count;
extern void init_isr(void);
extern void send_UART(unsigned int length);

/*************** hci.c ************************************/
extern BYTE send_data(void);                // send transmit_buffer with protocol length
extern BYTE HCI_Reset(void);
extern BYTE HCI_Read_Buffer_Size(void);
extern BYTE HCI_Write_Autentic_Enable(void);
extern BYTE HCI_Set_Event_Filter(void);
extern BYTE HCI_Write_Connection_Timeout(void);
extern BYTE HCI_Write_Page_Timeout(void);
extern BYTE HCI_Write_Scan_Enable(void);
extern BYTE HCI_Set_Event_Mask(void);
extern void CommandComplete(void);

/************* application.c ********************************/
```

```
extern bank1 unsigned int sample_time;
extern bank1 BYTE receive_SPI_on;
extern bank1 BYTE car_pwm;
extern bank1 BYTE timer_pwm_one;
extern bank1 BYTE timer_pwm_two;

void initControl(void);
void application_control(void);
void flashDiod(BYTE forever,int speed);
void error_SPI(void);

#endif
```

---------- Main.c ----------

```
#include <pic.h>
#include "global.h"

/****************** GLOBAL VARIABLE*********************************/
// Bank 0 local variable
// bank 1 global variable
// bank 2 global variable
// bank 3 incomming datapacket

bank1 system_reg_t system_reg;          // Control bits
bank1 unsigned int buffer_num;          // maximal number of package in buffer
bank1 unsigned int package_size;        // maximal number of byte in package
bank1 BYTE receive_buffer_index;        // possition in receive vector
bank1 BYTE transmit_buffer_index;       // position in receive vector
bank3 BYTE receive_buffer[80];          // Define receive_vector_1
bank2 BYTE transmit_buffer[20];         // Define transmit_vector
bank2 unsigned int package_length;      // length of receiving package
bank2 unsigned int transmit_length;     // length of transmitting package
bank2 BYTE connection_handle[2];        // connection_handle
bank2 BYTE digital_value;               // value of digital input
bank2 BYTE SPI_value;                   // value of SPI input
bank2 BYTE CTS;                         // RS-232
/*****************************************************************************/

/****************** DESCRIPTION ***************************************/
//The execution start in main and main calls initPIC
/*****************************************************************************/

//************************ WAIT ****************************************
void wait(unsigned int count){
        do{
        } while (count-- > 1);
}

//********************** INIT PIC ***************************************
static void initPIC(){

        // init control variable
        system_reg.bit_name.global_interrupt=0;
        system_reg.bit_name.transmit_allowed=1;
        system_reg.bit_name.timer_interrupt=0;
        system_reg.bit_name.receive_interrupt=0;
        system_reg.bit_name.spi_interrupt=0;
        system_reg.bit_name.spi_sending=1;
        system_reg.bit_name.digital_interrupt=0;
        system_reg.bit_name.init=0;             // 1 during init
        system_reg.bit_name.connected=0;
        system_reg.bit_name.new_data=0;

        // init variable
        buffer_num=1;
        package_size=255;
        receive_buffer_index=0;
        transmit_buffer_index=0;
        package_length=0;
        transmit_length=0;
        connection_handle[0]=0;
        connection_handle[1]=0;
```

```
        digital_value=0;


 //Configuration of inputs and outputs. Ports which are not used are set as outputs.
 TRISA=0x03;
 TRISB=0x10;                    // digital input initialiti disable
 TRISC=0xd0;

 RBPU=0;

// turn off diod on RC0
RC0=0;

//UART TRANSMITTER
SPBRG=0x03;
BRGH=1;
SYNC=0;
SPEN=1;
TX9=0;
TXIE=0;                         //Disable interrupt
TXEN = 1;

//UART RECEIVER
RCIE=1;
RX9=0;
CREN=1;

//SPI and SDO
SSPIE=0;                        // disable interrupt in SPI. Enable while receiving
STAT_SMP=1;
STAT_CKE=1;
CKP=0;
SSPM3=0;       //The following four bits set the speed of SPI synchronous speed to f_osc/4.
SSPM2=0;
SSPM1=0;
SSPM0=0;
SSPEN=1;

//Configuration of the digital input, which is the interrupt input RB0 on the PIC.
INTEDG=1;                       //Interrupt on rising slope
INTE=0;                         // disable interrupt

//ADconverter
//ADCON1
ADFM=1;
PCFG3=0;
PCFG2=1;
PCFG1=0;
PCFG0=0;

//ADCON0
ADCS1=0;
ADCS0=1;                        //DA-conversion to f_osc/8.
CHS2=0;
CHS1=0;
ADON=1;

//init timer0
T0CS=0;
T0IE=0;                         // disable interrupt
```

```
 // RTS/CTS
 RB3=1;                          //RTS=RB3
 CTS=RB4;

 //Clear all input to DA Converter
 RB2=0;
 wait(10);
 RB2=1;                          //not

 //Chip select DA
 RB1=1;                          // not

 //init PWM
 CCP1M3=1;
 CCP1M2=1;
 CCP2M3=1;
 CCP2M2=1;
 PR2=0x00;                       //period 0ms
 CCPR1L = 0x00;                  //cycle 0ms
 CCPR2L = 0x00;                  // cycle 0ms

 //init timer1 for PWM
 TMR1CS=0;                       //Use internat clock Fosc/4
 T1CKPS1=1;                      // timer1 prescale value is set to 8
 T1CKPS0=1;
 TMR1ON=0;                       //Timer1 off
 TMR1IE=0;                       //Disable interrupt

 //init timer2 for PWM
 TMR2ON=1;
 T2CKPS1=1;                       // timer2 prescale value is set to 16
 TMR2IE=0;                       //Disable interrupt

  //Enable interrupt
  GIE=1;
  PEIE=1;

 //sleep
  wait(200);


}

//************************ INIT BLUETOOTH*******************************
BYTE initBluetooth(void){

        BYTE temp;
        int data_temp;

        // init mode
        system_reg.bit_name.init = 1;

        temp = HCI_Reset();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Read_Buffer_Size();
```

```
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        // LSB transmitted first
        package_size = 0;
        package_size = (int) receive_buffer[7];
        data_temp = 0;
        data_temp = (int) receive_buffer[8];
        data_temp = data_temp<<8;
        package_size = package_size | data_temp;

        // LSB transmitted first
        buffer_num = 0;
        buffer_num = (int) receive_buffer[10];
        data_temp = 0;
        data_temp = (int) receive_buffer[11];
        data_temp = data_temp<<8;
        buffer_num = buffer_num | data_temp;

        temp = HCI_Write_Autentic_Enable();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Set_Event_Filter();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Write_Connection_Timeout();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Write_Page_Timeout();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Set_Event_Mask();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        temp = HCI_Write_Scan_Enable();
        if(temp != 0x00){                       // Command Error
                return temp;
        }

        system_reg.bit_name.init = 0;
        return 0x00;                            // init OK

}

/*****************************MAIN ************************************/
void main(void){

  BYTE success;
  BYTE number;
```

```
  initControl();
  init_isr();
  initPIC();

  // flash during one second. BT need one secod after reset
  flashDiod(0,5000);

  success = 0x01;
  number =0;

  while(success != 0x00){
        success = initBluetooth();

        // lasting error during init flash diod and wait for user reset
        if(number > 15){
          flashDiod(1,5000);
        }

        number++;
  }

  T0IE=1;          // enable interrupt
  application_control();

}
```

---------- Hci.c----------

```
#include <pic.h>
#include <string.h>
#include "global.h"

// Private variables
bank1 volatile static BYTE CommandCompleteResult = 0;

// ***************** CommandComplete *********************
/*
Called from interupt layer when a command complete event occurs
must NOT be called in any other situation.
*/
void CommandComplete(void){

        //Avoid stall
        if (receive_buffer[6] == 0xff){
                CommandCompleteResult = 0x01;
        } else {
                CommandCompleteResult = receive_buffer[6];
        }

}

// ****************** WaitForCommandCompete ****************
/*
All HCI command functions shall during init call this to wait for
command complete event from hostcontroller
*/
static void WaitForCommandComplete(void){

  // Wait on a spinlock until we receive a command complete event
  while(CommandCompleteResult == 0xFF) // Error replace 0xFF eith a define
  {
        wait(1);                // don't hang the computer
  }

}

//********************* Send Command *****************
static BYTE send_command(unsigned int length){


        CommandCompleteResult = 0xFF; // Engage the spinlock

        //allowed to transmit
        while (system_reg.bit_name.transmit_allowed == 0){
        }

        // Send the command
        send_UART(length);

        // Wait for command complete
        WaitForCommandComplete();
        // Return the result

        return CommandCompleteResult;
```

```
}

//****************** Send Data **********************
/*
My data package is only 21 byte so thay allways fit in
one bluetooth package. I haven't to bother about package_size
*/
BYTE send_data(void){

BYTE temp[2];

//flow control of RS-232, Bluetooth dosen't send correct CTS always 0.
if(CTS==0){

        // Bluetooth connected to other host
        if (system_reg.bit_name.connected == 1){

                //fix
                //buffer_num=10;

                //num of data package that fits in bluetooth buffer
                if (buffer_num > 0) {

                        //allowed to transmit
                        while (system_reg.bit_name.transmit_allowed == 0){
                        }

                        temp[0] = connection_handle[0];
                        temp[1] = (BYTE) (connection_handle[1] & 0x0f);
                        temp[1] = temp[1] | 0x20;

                        // HEAD of data package
                        transmit_buffer[0] = 0x02;//type
                        transmit_buffer[1] = temp[0];;
                        transmit_buffer[2] = temp[1];
                        transmit_buffer[3] = 0x0f; //length HCI LSB
                        transmit_buffer[4] = 0x00;
                        transmit_buffer[5] = 0x0b;//length L2CAP LSB
                        transmit_buffer[6] = 0x00;
                        transmit_buffer[7] = 0x00;//CID LSB
                        transmit_buffer[8] = 0x00;

                        transmit_length=20;

                        send_UART(transmit_length);
                        buffer_num--;

                        return 0x00;        //transmit OK

                        } else {
                                return 0x01;        //transmit error
                        }

                } else {
                        return 0x01;                //transmit error
                }

        } else {
                return 0x01;
        }
```

```
}


// ********************* HCI RESET ********************
BYTE HCI_Reset(void){

  static const BYTE data[] = {0x01,0x03, 0x0C, 0x00};

  transmit_length=4;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];

  return send_command(transmit_length);

}


// ******************** HCI_Read_Buffer_Size ************
BYTE HCI_Read_Buffer_Size(void){

  static const BYTE data[] = { 0x01,0x05,0x10,0x00 };

  transmit_length=4;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];

  return send_command(transmit_length);

}



// ****************** HCI_Write_Autentic_Enable **********
BYTE HCI_Write_Autentic_Enable(void){

  static const BYTE data[] = { 0x01,0x20,0x0c,0x01,0x00};

  transmit_length=5;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];

  return send_command(transmit_length);
}

// ***************** HCI_Set_Event_Filter **************
BYTE HCI_Set_Event_Filter(void){

  static const BYTE data[] = { 0x01,0x05,0x0c,0x03,0x02,0x00,0x02};

  transmit_length=7;
```

```
  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];
  transmit_buffer[5]=data[5];
  transmit_buffer[6]=data[6];

  return send_command(transmit_length);

}

// ********** HCI_Write_Connection_Timeout *********
BYTE HCI_Write_Connection_Timeout(void){

  static const BYTE data[] = { 0x01,0x16,0x0c,0x02,0xa0,0x1f};

  transmit_length=6;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];
  transmit_buffer[5]=data[5];

  return send_command(transmit_length);
}

// ********** HCI_Write_Page_Timeout **************
BYTE HCI_Write_Page_Timeout(void){

  static const BYTE data[] = { 0x01,0x18,0x0c,0x02,0x00,0x20};

  transmit_length=6;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];
  transmit_buffer[5]=data[5];

  return send_command(transmit_length);
}

// ********** HCI_Write_Scan_Enable ***************
BYTE HCI_Write_Scan_Enable(void){

  static const BYTE data[] = { 0x01,0x1a,0x0c,0x01,0x03};

  transmit_length=5;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];
```

```
  return send_command(transmit_length);
}

//***************** HCI_Set_Event_Mask ******************
BYTE HCI_Set_Event_Mask(void){

  static const BYTE data[] = { 0x01,0x01,0x0c,0x08,0x14,0x60,0x04,0x00,0x00,0x00,0x00,0x00};

  transmit_length=12;

  transmit_buffer[0]=data[0];
  transmit_buffer[1]=data[1];
  transmit_buffer[2]=data[2];
  transmit_buffer[3]=data[3];
  transmit_buffer[4]=data[4];
  transmit_buffer[5]=data[5];
  transmit_buffer[6]=data[6];
  transmit_buffer[7]=data[7];
  transmit_buffer[8]=data[8];
  transmit_buffer[9]=data[9];
  transmit_buffer[10]=data[10];
  transmit_buffer[11]=data[11];

  return send_command(transmit_length);

}
```

```
---------------------------------------------- Isr.c ----------------------------------------------

#include <pic.h>
#include "global.h"

bank1 static int time_count;
bank1 static BYTE first;
bank1 static BYTE error;
bank1 BYTE car_count;

/********************** Description ***********************************/


/**************************************************************************/

void init_isr(void){

        time_count=0;
        car_count=1;
        first=1;
        error=0;
}


/************************** ISR ***************************************/
interrupt isr(void){

        int t;                  //testing
        int data_temp;

        /* TIMER0 GENERALL INTERRUPT*/
        if(T0IF == 1){

                //Sample of input
                if(time_count >= sample_time){

                        time_count=0;
                        system_reg.bit_name.timer_interrupt=1;
                        system_reg.bit_name.global_interrupt=1;

                } else {
                        time_count++;
                }

                //Simulated PWM
                if (car_pwm == 0x01){

                        if(car_count >= 80){

                                car_count=0;

                                //PWM1
                                if(timer_pwm_one != 0x00){

                                        RC2=1;                  // set high value on PWM
                                        TMR1ON=1;               //enable timer

                                        //set timer register
                                        TMR1H=0xff;     //timer 1 equal with TMR1L prescaled 16
                                        TMR1L=(0xff-timer_pwm_one);
```

```
                        }

                        //PWM2
                        if(timer_pwm_two !=0x00){

                                //set timer register
                                PR2=timer_pwm_two;          //timer 2 prescaled 16
                                RC1=1;                      // set high value on PWM
                                TMR2ON=1;                   //enable timer
                        }

                } else {
                        car_count++;
                }
        }

        T0IF=0;
}

/* TIMER1 INTERRUPT PWM1*/
if(TMR1IF == 1){

        if(car_pwm == 0x01){
                RC2=0;
                TMR1ON=0;       //Stop timer
        }

        TMR1IF=0;
}



/* TIMER2 INTERRUPT PWM2*/
if(TMR2IF == 1){

        if(car_pwm == 0x01){
                RC1=0;
                TMR2ON=0;       //Stop timer
        }

        TMR2IF=0;
}



/* SPI INTERRUPT RECEIVING MODE*/
if (system_reg.bit_name.spi_sending==0 && SSPIF==1) {

        // Error last event not handled avoid losing information
        if(system_reg.bit_name.spi_interrupt==1){
                error_SPI();
        }

        SPI_value=SSPBUF;
        system_reg.bit_name.spi_interrupt = 1;
        system_reg.bit_name.global_interrupt = 1;

        SSPIF=0;
```

```
}


/* DIGITAL INTERRUPT */
if(INTF==1){

        //No interrupt during powerup and init
        if(system_reg.bit_name.init == 0){

                // old interrupt not handled avoid deadlock on SPI
                if (system_reg.bit_name.digital_interrupt == 1 ){

                        // if two interrupts occure simultaniously, before the first was handled,
                        // is it the same as no interrupt at all arrived.
                        system_reg.bit_name.digital_interrupt=0;

                } else {

                        digital_value = RB0;    //RB0 == 1 of course
                        system_reg.bit_name.digital_interrupt=1;
                        system_reg.bit_name.global_interrupt=1;
                }
        }

        INTF=0;
}


/* RECEIVE UART INTERRUPT */
if (RCIF){

        receive_buffer[receive_buffer_index++] = RCREG;

        // Garbage byte at position zero. Allways a zero on reset
if(receive_buffer_index == 1 && (receive_buffer[0] != 0x02 && receive_buffer[0] != 0x04) ){
                receive_buffer_index--;
        }

        // Two byte large FIFO receiving buffer
        if(RCIF){
                receive_buffer[receive_buffer_index++] = RCREG;

                // Garbage byte at position zero
if(receive_buffer_index == 1 && (receive_buffer[0] != 0x02 && receive_buffer[0] != 0x04) ){
                        receive_buffer_index--;
                 }

        }

        // Error receiving, to large packet. Start looking after data or event packages
        if(receive_buffer_index >= 30){
                receive_buffer_index=0;
        }

        //event package length
        if(((receive_buffer_index -1) >= 2) && (receive_buffer[0] == 0x04) && (first==1)){

                package_length=(int) receive_buffer[2];
```

```
                first = 0;
        }

        //data package length
        if(((receive_buffer_index - 1) >= 4) && (receive_buffer[0] == 0x02) && (first==1)){

                package_length = (int) receive_buffer[3];
                data_temp=0;
                data_temp = receive_buffer[4];
                data_temp = data_temp<<8;
                package_length = package_length | data_temp;
                first = 0;
        }

        //synchronization to keep data structure intact. First data byte read
        if(receive_buffer[0] == 0x02 && receive_buffer_index == 10){
                system_reg.bit_name.new_data=1;
        }

        //received whole event package
        if(receive_buffer[0] == 0x04 && receive_buffer_index == (package_length+3)){

                receive_buffer_index=0;
                package_length=0;
                first=1;

                //init of bluetooth
                if(system_reg.bit_name.init == 1){
                        CommandComplete();
                } else {
                        system_reg.bit_name.receive_interrupt=1;
                        system_reg.bit_name.global_interrupt=1;
                }

        }

        //received whole data package
        if(receive_buffer[0] == 0x02 && receive_buffer_index == (package_length+5)){
                receive_buffer_index=0;
                package_length=0;
                first=1;
                //synchronization to keep data structure intact
                system_reg.bit_name.new_data=0;
                system_reg.bit_name.receive_interrupt=1;
                system_reg.bit_name.global_interrupt=1;
        }

}



/* Error receiving cleared in software*/
if (OERR==1){
        OERR=0;
        CREN=0;
CREN=1;
}

}
```

```
void send_UART(unsigned int length) {

        transmit_buffer_index = 0;
        system_reg.bit_name.transmit_allowed=0;

        while(transmit_buffer_index < length){

                TXREG = transmit_buffer[transmit_buffer_index++];

                //transmit buffer empty
                while(TRMT == 0){
                }

        }

        system_reg.bit_name.transmit_allowed = 1;

}
```

```
----------------------------------------------------------Application.c----------------------------------------------------------
#include <pic.h>
#include "global.h"

/***************** DESCRIPTION ***********************************/

/*****************************************************************/

bank1 static unsigned int pos_SPI_out;
bank1 BYTE car_pwm;
bank1 unsigned int sample_time;
bank1 BYTE receive_SPI_on;
bank1 BYTE spi_data[6];
bank1 BYTE int_value;
bank1 BYTE timer_pwm_one;
bank1 BYTE timer_pwm_two;

//************************ INIT APPLICATION*********************************
void initControl(void){
  pos_SPI_out = 0;

  // 5 ggr/s
  sample_time=800;

  // Not receiving
  receive_SPI_on=0;

  // using PIC PWM
  car_pwm=0x00;         //false

  //Setting digital interrupt to 0 between interrupts
  int_value=0;

}
// ************************ flash diod ****************************
// reset = flash one second forever=0
// error = flash forever seconds forever=1
void flashDiod(BYTE forever,int speed){

  int t;
  int stop;

  if (forever==1){

            while(1){

              RC0=1;
              for(t=0;t<speed;t++){
              }

              RC0=0;
              for(t=0;t<speed;t++){
              }

            }

  } else {

            stop=0;
            while(stop<10){
```

```
            RC0=1;
            for(t=0;t<speed;t++){
            }

            RC0=0;
            for(t=0;t<speed;t++){
            }

            stop++;
          }
  }

}

//************************ INAD *********************************
// Sample analog input and store values in transmit_buffer at pos 9-12
static void inAD(void){

   //Storing the voltage from the FIRST analog input.
   CHS0=0;
   wait(5);
   ADGO=1;
   while(ADGO){
   };
   transmit_buffer[10] = ADRESH;
   transmit_buffer[10] = (BYTE) (transmit_buffer[10] & 0x03);          //Only two LSB
   transmit_buffer[9] = ADRESL;                                        // LSB

   //Storing the voltage from the SECOND analog input.
   CHS0=1;
   wait(5);
   ADGO=1;
   while(ADGO){
   };
   transmit_buffer[12] = ADRESH;
   transmit_buffer[12] = (BYTE) (transmit_buffer[12] & 0x03);          //Only two MSB
   transmit_buffer[11] = ADRESL;                                       // LSB
}

//******************** Output to DA ****************************
// Write Analog output from received_buffer to the cards DA converter
static int outAnalog(void){

  BYTE temp;
  BYTE data[4];

  temp=0x00;

  //synchronization to keep data structure intact
  if(system_reg.bit_name.new_data == 0){
          data[0] = receive_buffer[9];
          data[1] = receive_buffer[10];
          data[2] = receive_buffer[11];
          data[3] = receive_buffer[12];
  } else {
          return 1;
  }

  if(system_reg.bit_name.new_data == 0){
```

```
//Send to DA A and update its input register
RB1=0;                          // CS
temp = 0x20;
temp = temp | (data[1] << 3);
temp = temp | (data[0] >> 5);

SSPBUF=temp;

temp=0x00;
temp = temp | (data[0]<<3);

while(SSPIF==0){
}
SSPIF=0;

SSPBUF=temp;

while(SSPIF==0){
}
SSPIF=0;

RB1=1;

//Send to DA B and update its input register
RB1=0;              // CS
temp = 0xa0;
temp = temp | (data[3] << 3);
temp = temp | (data[2] >> 5);

SSPBUF=temp;

temp=0x00;
temp = temp | (data[2]<<3);

while(SSPIF==0){
}
SSPIF=0;

SSPBUF=temp;

while(SSPIF==0){
}
SSPIF=0;

RB1=1;

//Send to DA and output its input register
RB1=0;              // CS
SSPBUF=0x80;

while(SSPIF==0){
}
SSPIF=0;

SSPBUF=0x00;

while(SSPIF==0){
}
SSPIF=0;
```

```
        RB1=1;

        return 0;

  } else {
        return 1;
  }
}

// ********************** STORE_SPI ****************************
// store SPI byte. Send by storing in transmit_buffer when received all bytes;
static void store_SPI(void){

        // store
        spi_data[pos_SPI_out++]=SPI_value;

        //read all 6 byte
        if (pos_SPI_out == 6){

                transmit_buffer[14]=spi_data[0];
                transmit_buffer[15]=spi_data[1];
                transmit_buffer[16]=spi_data[2];
                transmit_buffer[17]=spi_data[3];
                transmit_buffer[18]=spi_data[4];
                transmit_buffer[19]=spi_data[5];

                pos_SPI_out=0;
        }


}

// ********************** ERROR_SPI*********************************
// reciving a byte when not done with the old interrupt
void error_SPI(void){

        // Store the old not handled value in right position.
        store_SPI();
}

// ************************ STORE_DIGITAL ************************
// store digital input in transmit_buffer place 13;
static void store_DIGITAL(void){

        transmit_buffer[13] = digital_value;
        int_value=1;

}

// ********************** STORE_CTS ****************************
// store flow control from RS-232
static void store_CTS(void){

        CTS=RB4;
}

// ************************ OUT PWM ****************************
// Set PWM in PIC
static int outPWM(void){
```

```
        BYTE temp;
        BYTE data[5];

        //synchronization to keep data structure intact
        if(system_reg.bit_name.new_data == 0){
          data[0]=receive_buffer[13];
          data[1]=receive_buffer[14];
          data[2]=receive_buffer[15];
          data[3]=receive_buffer[16];
          data[4]=receive_buffer[17];
        } else {
          return 1;
        }

        if(system_reg.bit_name.new_data == 0){

                PR2=data[0];                    //period

                //duty cycle one
                temp = (BYTE) (data[2]<<6);
                temp = temp | (data[1]>>2);
                CCPR1L = temp;
                temp = (BYTE) (data[1] & 0x03);
                CCP1X = (temp>>1);
                CCP1Y = (BYTE) (data[1] & 0x01);

                //duty cycle two
                temp = (BYTE) (data[4]<<6);
                temp = temp | (data[3]>>2);
                CCPR2L = temp;
                temp = (BYTE) (data[3] & 0x03);
                CCP2X = (temp>>1);
                CCP2Y = (BYTE) (data[3] & 0x01);

                return 0;
        } else {
                return 1;
        }

}

// ****************** rampDown *****************************
// Ramp down signal when losing connection
static void rampDown(void){

  int t;

  // set analog outputs to zero

  //Send to DA A and update its input register
  RB1=0;                                // CS

  SSPBUF=0x20;

  while(SSPIF==0){
  }
  SSPIF=0;

  SSPBUF=0x00;
```

```
while(SSPIF==0){
}
SSPIF=0;

RB1=1;

//Send to DA B and update its input register
RB1=0;                  // CS
SSPBUF=0xa0;

while(SSPIF==0){
}
SSPIF=0;

SSPBUF=0x00;

while(SSPIF==0){
}
SSPIF=0;

RB1=1;

//Send to DA and output its input register
RB1=0;                  // CS
SSPBUF=0x80;

while(SSPIF==0){
}
SSPIF=0;

SSPBUF=0x00;

while(SSPIF==0){
}
SSPIF=0;

RB1=1;


//if car_pwm change to PIC PWM
if(car_pwm == 0x01){

        car_pwm = 0x00;

        //turn off interrupt
        TMR1IE=0;
        TMR2IE=0;

        //turn off timer1
        TMR1ON=0;

        //Enable PIC PWM
        CCP1M3=1;
        CCP1M2=1;
        CCP1M1=1;
        CCP1M0=1;

        CCP2M3=1;
        CCP2M2=1;
        CCP2M1=1;
```

```
        CCP2M0=1;

        // enable timer2
        TMR2ON=1;
 }

 // set PWM to zero
 PR2=0xff;                  //period

 //duty cycle one
 CCPR1L = 0x00;
 CCP1X = 0;
 CCP1Y = 0;

 //duty cycle two
 CCPR2L = 0x00;
 CCP2X=0;
 CCP2Y =0;

 //set TransmittBuffer to zero
 for(t=0;t<20;t++){
        transmit_buffer[t]=0x00;
 }

}

//********************** APPLICATION_CONTROL**********************
/*
Main loop
*/
void application_control(void){

        BYTE more_interrupts;
        int temp;
        int data_temp;

        while(1){

                //wait for interrupt
                while(system_reg.bit_name.global_interrupt==0){
                }

                more_interrupts=1;  //TRUE

                // deal with interrupt
                while(more_interrupts){



                        /* RECEIVE PACKAGE FROM BLUETOOTH */
                        if(system_reg.bit_name.receive_interrupt == 1){

                                //All allowed event package
                                if(receive_buffer[0] == 0x04){

                                   /* Connection complete event */
                                   if(receive_buffer[1] ==0x03 && receive_buffer[3]==0x00){

                                        connection_handle[0] = receive_buffer[4];
                                        connection_handle[1] = (BYTE) (receive_buffer[5] & 0x0f);
```

```
                    // connected to other bluetooth host
                    system_reg.bit_name.connected = 1;

                    // turn on diod
                    RC0=1;
                }


                /* Disconnect complete event*/
                if(receive_buffer[1] ==0x05 && receive_buffer[3] == 0x00){

                    system_reg.bit_name.connected=0;
                    connection_handle[0]=0;
                    connection_handle[1]=0;

                    rampDown();

                    // Bluetooth data buffer
                    buffer_num = 10;

                    sample_time=1000;

                    // turn off diod
                    RC0=0;

                }

                /* Number of complete package*/
                if(receive_buffer[1] ==0x13){

                    temp = 0;
                    temp = (int) receive_buffer[6];
                    data_temp=0;
                    data_temp=receive_buffer[7];
                    data_temp=data_temp<<8;
                    temp = temp | data_temp;

                    buffer_num += temp;

                }

                /* Command Complete Event*/
                if(receive_buffer[1] ==0x0e && receive_buffer[6]==0x00){
                }

                /* Command Status Event*/
                if(receive_buffer[1] ==0x0f && receive_buffer[3]==0x00){
                }


            // data package
            } else if(receive_buffer[0] == 0x02){

                //Store period time for simulated PWM
                timer_pwm_one=receive_buffer[14];
                timer_pwm_two=receive_buffer[16];

                //set Analog outputs if in SPI sending mode
                if(system_reg.bit_name.spi_sending == 1){
```

```
            outAnalog();
}

//PIC PWM
if(car_pwm == 0x00 && receive_buffer[13] != 0xff){

            //set PWM
            outPWM();

// Change from simulated PWM to PIC PWM.
} else if(car_pwm ==0x01 && receive_buffer[13] != 0xff){

            car_pwm = 0x00;

            //turn off interrupt
            TMR1IE=0;
            TMR2IE=0;

            //turn off timer1
            TMR1ON=0;

            // enable timer2
            TMR2ON=1;

            //Enable PIC PWM
            CCP1M3=1;
            CCP1M2=1;
            CCP1M1=1;
            CCP1M0=1;

            CCP2M3=1;
            CCP2M2=1;
            CCP2M1=1;
            CCP2M0=1;

            //setPWM
            outPWM();

// Change from PIC PWM to simulated PWM
} else if(car_pwm == 0x00 && receive_buffer[13] == 0xff){

            //Disable PIC PWM
            CCP1M3=0;
            CCP1M2=0;
            CCP1M1=0;
            CCP1M0=0;

            CCP2M3=0;
            CCP2M2=0;
            CCP2M1=0;
            CCP2M0=0;

            //variabel in isr
            car_count=1;

            //stop timers to get synchronized start
            TMR1ON=0;
            TMR2ON=0;

            //setoutput
```

```
                RC1=0;
                RC2=0;

                //enable interrupt on timer1 and timer2
                TMR1IE=1;
                TMR2IE=1;

                // first change of period
                car_pwm = 0x01;

        //simulated PWM

        } else if(car_pwm ==0x01 && receive_buffer[13] == 0xff){

                // The system handles simulated PWM by timers
        }

        // set sample period of analog inputs
        if (receive_buffer[18] > 4){
                temp=0;
                temp = (int) receive_buffer[18];
                sample_time = temp*4;
        } else {
                // min  4 ms
                sample_time=16;
        }

        // enable/disable SPI intput
        if (receive_SPI_on==0){

                //enable otherwise continue
                if( receive_buffer[19]==1){

                   TRISB0=1;
                   pos_SPI_out = 0;
                   receive_SPI_on=1;

                   wait(5);
                   INTE=1;                    // enable interrupt
                }


        } else {

                //disable otherwise continue
                if( receive_buffer[19]==0){

                   //Allways allow AD conversion
                   if(system_reg.bit_name.spi_sending==0){


                         //disable SPI interrupt
                         SSPIE=0;
                         system_reg.bit_name.spi_sending = 1;
                   }

                    TRISB0=0;
                    receive_SPI_on=0;

                   wait(5);
```

```
                                INTE=0;                 // disable interrupt
                            }
                        }


                }

                system_reg.bit_name.receive_interrupt = 0;

        /* INTERRUPT ON DIGITAL INPUT */
        } else if(system_reg.bit_name.digital_interrupt == 1){

                //Possible chance of deadlock error if digital interrupt to close!!!
                system_reg.bit_name.digital_interrupt = 0;

                // SPI receiving mode enable
                if(receive_SPI_on==1){

                        store_DIGITAL();

                // Reveive on SPI Recive data change between receiving and sending
                        if(system_reg.bit_name.spi_sending == 1){

                                //Chip not select DA
                                RB1=1;

                                //Enable SPI interrupt
                                SSPIE=1;

                                pos_SPI_out = 0;

                                system_reg.bit_name.spi_sending = 0;

                        // Send on SPI
                        } else {

                                //disable SPI interrupt
                                SSPIE=0;

                                system_reg.bit_name.spi_sending = 1;
                        }

                }


        /* INTERRUPT ON SPI */
        } else if(system_reg.bit_name.spi_interrupt == 1){

                // receiving spi
                if(system_reg.bit_name.spi_sending == 0){

                        // Store data in transmit vector
                        store_SPI();

                }

                system_reg.bit_name.spi_interrupt = 0;
```

```
                        /* TIMER INTERRUPT */
                        } else if(system_reg.bit_name.timer_interrupt == 1){

                                //Reseting int interrupt value after detection. Interrupt only on ones
                                if(int_value >= 2){
                                        transmit_buffer[13]=0x00;
                                        int_value=0;
                                } else if(int_value == 1) {  //always send the detected interrupt
                                        int_value++;
                                }

                                inAD();
                                store_CTS();

                                //Send data to bluetooth if we are connected
                                send_data();

                                system_reg.bit_name.timer_interrupt = 0;

                        }



                        // more interrupt available while dealing with this
                        if(     system_reg.bit_name.timer_interrupt == 0 &&
                                system_reg.bit_name.receive_interrupt == 0 &&
                                system_reg.bit_name.spi_interrupt == 0 &&
                                system_reg.bit_name.digital_interrupt == 0){

                                more_interrupts=0;    //FALSE
                                system_reg.bit_name.global_interrupt=0;
                        }

                }


        }

}
```

Appendix D: Code in master

---------- ControllLayer ----------

```
import java.util.Vector;

public class ControllLayer {

 private L2CAP capLayer;
 private HCI hciLayer;
 private API application;

 private int numOfCommandPackage;
 private int initState;
 private int flowNum;
 private int flowSize;
 private boolean debug;

 private boolean sucess;                    // return value
 private boolean init;                      // True -> the stack is initialised
 private Host temp;
 private HostList myList;
 private String myPort;

 /**
 * Constructor
 *
 * @param String comPort the port COM the stack will use
 * @param topLayer the next layer in the stack.
 */
 public ControllLayer(String comPort,API topLayer){

          myList = new HostList();
          hciLayer = new HCI(this);
         capLayer = new L2CAP(1,320,this,hciLayer);
         RS232 r = new RS232(comPort,hciLayer,this);
         application=topLayer;
         myPort=comPort;

         hciLayer.setInitValue(capLayer,r);

         numOfCommandPackage=1;
         initState=0;
         sucess = false;
         init= false;
         debug=true;
 }

 /****************************** Public methods for API ************************************/

  /**
 * Initialise Bluetooth controller. The Thread is blocked until success or failure
 *
 * @return true if success otherwise false
 */
 public synchronized  boolean init(){

         try {

                        initState=0;
```

```
                        myInit();
                        wait();
                        return sucess;

        } catch (Exception e){

                write("E -> Exception during init " + e.toString() + " ");
                return false;
        }
}

/**
* Inquire after other Bluetooth devices. The Thread is blocked until success or failure
*
* @return true if success otherwise false
*/
public synchronized boolean inquiry(){

        try{

                // must have initialised the connection before inquiry
                if (init){
                        byte data[] = new byte[9];
                        data[1] = 0x01;
                        data[2] = 0x04;
                        data[3] = 0x05;
                        data[4] = 0x33;
                        data[5] = (byte) 0x8b;
                        data[6] = (byte) 0x9e;
                        data[7] = 0x05;
                        data[8] = 0x00;

                        hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);

                        //blocking
                        wait();
                        return sucess;
                } else {
                        return false;
                }

        } catch (Exception e){

                write("E -> Exception during inquiry " + e.toString()+ " ");
                return false;
 }
}

/**
* Connect to Bluetooth device. The Thread is blocked until success or failry
*
* @param int Bluetooth address
* @return the CID of L2CAP layer 0x0000 if error
*/
public synchronized byte[] createConnection(byte[] adress){

        Host temp;
        byte data[];
        byte myCID[];
        byte tempv[];
```

```
tempv = new byte[2];
myCID = new byte[2];

if(numOfCommandPackage >0 && init){

        temp = myList.getHost(adress);

        if (temp != null){
                byte[] clock;

                // The bluetooth host is already connected
                if (temp.getConnected()){
                        write("I -> Can't connect the bluetooth host is already connected ");
                return tempv;
                }

                ata = new byte[17];
                lock=temp.getClockOffset();

                data[1] = 0x05;
                data[2] = 0x04;
                data[3] = 0x0d;
                data[4] = adress[0];          //MSB
                data[5] = adress[1];
                data[6] = adress[2];
                data[7] = adress[3];
                data[8] = adress[4];
                data[9] = adress[5];
                data[10] = 0x08;
                data[11] = 0x00;
                data[12] = temp.getScanRepetition();
                data[13] = temp.getScanMode();
                data[14] = clock[0];          //MSB
                data[15] = clock[1];
                data[16] = 0x00;

                //buffer empty when not connected. Needed when reconecting lost connection
                capLayer.setFlowControl(flowNum,flowSize);

                hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);

                //Only one connection and therefore only one CID
                myCID[0] = 0x01;
                myCID[1] = 0x00;

                temp.setCID(myCID);

        try {

                //blocking
                wait();

         } catch (Exception e){

                write("E -> Exception during create connection "  + e.toString() + " ");
                return tempv;
        }

        if (sucess){
```

```
                        return myCID;
                } else {
                        return tempv;
                }

        } else {
                write("E -> Error connecting to bluetooth, adress not valid ");
                return tempv;  // 0x0000;
        }

  } else {
        return tempv;                   // 0x0000;
  }

}

/**
* Send data packet.
*
* @param char[] CID number
* @param char[] The data
*/
public void sendData(byte[] CID,byte[] data){

        Host temp;
        byte[] test;
        byte[] dataPackage;
        byte[] newHandle;
        byte[] tempHandle;
        temp = null;
        newHandle = new byte[2];


        //find bluetooth connection
        for (int i=0;i<myList.getSize();i++){
                temp = (Host) myList.getHostAt(i);
                test = temp.getCID();
                if (test[0] == CID[0] && test[1]==CID[1]){
                        break;
                }
        }

        if(temp != null && temp.getConnected()){

                dataPackage = new byte[20];
                for (int i =9;i<dataPackage.length;i++){
                        dataPackage[i]=data[i-9];
                }
                tempHandle = temp.getConnectionHandle();

                newHandle[0]=tempHandle[0];
                newHandle[1]=tempHandle[1];

                capLayer.sendACL(temp.getCID(),newHandle,dataPackage);

        } else {
                write("E -> Bluetooth device not connected. Error sending data ");
        }

}
```

```
/**
* Close Connection
*
* @param byte[] the Bluetooth adress of the external host
*/
public void closeConnection(byte[] btAdress){

	byte[] data;
	byte[] handle;
	Host temp;

	if(numOfCommandPackage>0){

		data= new byte[7];

		temp=myList.getHost(btAdress);

		if (temp != null && temp.getConnected() == false){
			write("E -> Can't disconnect. The host is not connected ");
		}

		if(temp != null){

			handle=temp.getConnectionHandle();

			data[1]=0x06;
			data[2]=0x04;
			data[3]=0x03;
			data[4]=handle[0];
			data[5]=handle[1];
			data[6]=0x13;			//Reason

			hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);

		} else {

			writeByte("E -> Error disconnecting bluetooth host, CID not valid ", btAdress);
		}

	} else {
		write("E -> Cant disconnect. Bluetooth command buffer is full ");
	}
}

/*
* Read the RSSI value, strength of connection, of the host. The Bluetooth module must have
* an connection to send a readRSSI request
*
* @param byte[] The address of the Bluetooth device
*/
public void readRSSI(byte[] btAdress){

	byte[] data;
	byte[] handle;
	Host temp;

	if(numOfCommandPackage>0){

		data = new byte[6];
```

```
                temp=myList.getHost(btAdress);

                if (temp != null && temp.getConnected() == false){
                        writeByte("E -> Can't read RSSI. Host not connected or invalid host ",btAdress);
                } else {

                        handle=temp.getConnectionHandle();

                        data[1]=0x05;
                        data[2]=0x14;
                        data[3]=0x02;
                        data[4]=handle[0];
                        data[5]=handle[1];

                        hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);

                }

        } else {
                write("E -> Can't send command package. Bluetooth buffer is full ");
        }
}

/*
* Software reset of Bluetooth device
*
 */
public void reset(){

        byte[] data = new byte[4];

        data[1]=0x03;
        data[2]=0x0c;
        data[3]=0x00;

        if (numOfCommandPackage >0){

                hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);
        } else {
                write("I -> Can't send reset package. Bluetooth buffer is full ");
        }
}


 /****************** Public method available for lower parts of the stack **********************/

 /**
 * Receive data packet
 *
 * @param String The data
 * @param int the position of the first data byte
 */
 public void receiveACL(byte[] data,int pos){

        //writeByte(data,pos,"Receiving data: ");
        application.receiveData(data,pos);
 }

 /**
```

```
* Receive event
*
* @param The event
* @param int the index of the first byte
*/
public synchronized void receiveEvent(byte[] s,int pos){

	Host temp;
	int eventCode;
	byte adress[];
	String inData="";

	adress = new byte[6];

	temp=null;

	for(int i=0;i<s.length;i++){
		inData=inData + "0x" + UnicodeFormatter.byteToHex(s[i]) + " ";
	}

	write("I -> Incomming event " + inData);

	eventCode = (int) s[pos];

	switch (eventCode) {

		/* Inquiry Complete Event */
		case 1 :  // Inquiry OK
			if(s[2+pos] == 0x00){

				if(myList.getSize() == 0){
					write("E -> No bluetooth hosts answering ");
					sucess = false;
					releaseLock();

				} else {

					//change status of Host and notify application
					for (int i=0;i<myList.getSize();i++){
						temp = (Host) myList.getHostAt(i);
						if (temp.getStatus() == false){
							temp.setStatus(true);
						}
					}

					sucess = true;
					releaseLock();
				}

			} else {

				// remove Host from system
				for (int i=0;i<myList.getSize();i++){
					temp = (Host) myList.getHostAt(i);
					if (temp.getStatus() == false){
						myList.removeHost(temp);
					}
				}

				sucess = false;
```

```
                releaseLock();

                write("E -> Inquiry failed ");
            }
            break;

        /* Inquiry Result Event */
        case 2 :  byte[] clock;

            clock = new byte[2];

            adress[0]= s[3+pos];
            adress[1]= s[4+pos];
            adress[2]= s[5+pos];
            adress[3]= s[6+pos];
            adress[4]= s[7+pos];
            adress[5]= s[8+pos];

            clock[0] =s[15+pos];
            clock[1] =s[16+pos];

            temp = new Host(adress,s[9+pos],s[10+pos],s[11+pos],clock);

            myList.storeHost(temp);

            // notify application
            application.BTFound(temp.getAdress());

            if(s[2+pos] != 0x01){
                write("I -> More then one inquiry result, bother only of the first ");
            }
            break;

        /* Connection complete event */
        case 3 :  //Command ok
            if(s[2+pos] ==0x00){

                byte[] connectionHandle;

                connectionHandle = new byte[2];

                connectionHandle[0]=s[3+pos];
                connectionHandle[1]= (byte) (s[4+pos] & 0x0f);


                adress[0]=s[5+pos];
                adress[1]=s[6+pos];
                adress[2]=s[7+pos];
                adress[3]=s[8+pos];
                adress[4]=s[9+pos];
                adress[5]=s[10+pos];

                // insert connectionHandle
                temp = myList.getHost(adress);
                temp.setConnectionHandle(connectionHandle);
                temp.setConnected(true);

                sucess = true;
                releaseLock();
```

```
		} else {

			sucess = false;
			releaseLock();

			write("E -> Error code connection complete event ");
		}
		break;

	/* Disconnection Complete Event*/
	case 5 :  //Command ok
		if(s[2+pos] ==0x00){

			byte[] handle;
			byte[] tempHandle;
			boolean found=false;

			handle = new byte[2];
			tempHandle = new byte[2];

			handle[0]=s[3+pos];
			handle[1]= (byte) (s[4+pos] & 0x0f);

			//remove connectionHandle in host
			for (int i=0;i<myList.getSize();i++){
			temp = (Host) myList.getHostAt(i);
			if (BlackBox.equals(handle,temp.getConnectionHandle())){
				if(temp.getConnected() == true){

					//write("I -> Host disconnect");
					tempHandle[0]=0;
					tempHandle[1]=0;
					temp.setConnected(false);
					temp.setConnectionHandle(tempHandle);
					application.connectionIsClosed(temp.getAdress());
					found=true;

					break;

				} else {
					write("E -> The host is not connected ");
				}
			}
			}

			if(!found){
		write("E -> Disconnection failed in bluetooth stack. Bluetooth host not found ");
			}

		} else {
			write("E -> Error code disconnection complete event ");
		}
		break;

	/* Command Complete Event */
	case 14 :// command ok
		if(s[5+pos] == 0x00){

			// save commandpackage buffer
			numOfCommandPackage = (int) s[2+pos];
```

```
                    //Read Buffer Size
                    if (s[3+pos] == 0x05 && s[4+pos] == 0x10){
                            int num;
                            int size;

                            size=s[6+pos];
                            size += s[7+pos]<<8;

                            num=s[9+pos];
                            num += s[10+pos]<<8;

                            flowNum=num;
                            flowSize=size;

                            capLayer.setFlowControl(num,size);
                    }

                    // RSSI
                    if(s[3+pos]==0x05 && s[4+pos] == 0x14){

                            //only one connection
                            application.receiveRSSI(s[pos+8]);

                    }

                    // Next initialisation step
                    if (initState <=5){
                            myInit();
                    } else {

                            // init suceeded
                            sucess = true;
                            init=true;
                            releaseLock();
                    }

            } else {

                    // Initialisation failed
                    sucess = false;
                    init = false;
                    releaseLock();

                    write("E -> Error code commandcomplete event ");
            }
            break;

    /* Command status event */
    case 15 ://Command Ok
            if(s[2+pos] ==0x00){
                    numOfCommandPackage = (int) s[3+pos];

            } else {
                    write("E -> Error code command status event ");
            }
            break;


    /* number of complete packets event*/
```

```
                case 19 :       if(s[2+pos] == 0x01){

                                        int sendPackage;

                                        sendPackage=s[5+pos];
                                        sendPackage += s[6+pos]<<8;

                                        capLayer.finishedSendPackage(sendPackage);

                                } else {

                                write("E -> More then one connectionhandle in complete package event"
                                        + ",bother only of the first. ");

                                }
                                break;


                default : application.receiveEvent(s);

        }
}


/*********************************Private methods ***********************************/

/**
* Intern initialisation of Bluetooth Controller
*/
private synchronized void myInit(){

        byte data[];

        // Compiler complains
        data = new byte[1];

        // In case of multiple initialisation threads
        switch(initState){

                // HCI Reset
                case 0 :  data = new byte[4];

                        data[1]=0x03;
                        data[2]=0x0c;
                        data[3]=0x00;
                        break;

                // HCI Read buffer Size
                case 1 :  data = new byte[4];
                        data[1]=0x05;
                        data[2]=0x10;
                        data[3]=0x00;
                        break;

                // HCI Write Autentic Enable
                case 2 :  data = new byte[5];
                        data[1]=0x20;
                        data[2]=0x0c;
                        data[3]=0x01;
                        data[4]=0x00;
```

```
                        break;

                // HCI Set Event Filter
                case 3 :  data = new byte[7];
                        data[1]=0x05;
                        data[2]=0x0c;
                        data[3]=0x03;
                        data[4]=0x02;
                        data[5]=0x00;
                        data[6]=0x02;
                        break;

                // HCI Write Connection Accept Timeout
                case 4 :  data = new byte[6];
                        data[1]=0x16;
                        data[2]=0x0c;
                        data[3]=0x02;
                        data[4]=(byte) 0xa0;
                        data[5]=0x1f;
                        break;

                // HCI Write Page Timeout
                case 5:   data = new byte[6];
                        data[1]=0x18;
                        data[2]=0x0c;
                        data[3]=0x02;
                        data[4]=(byte) 0x00;
                        data[5]=0x20;
                        break;

                        default : write("E -> Error init bluetooth ");
                }


        // Am I allowed to send data package
        if(numOfCommandPackage >0){
                initState++;
                hciLayer.sendData(data,BlackBox.COMMANDPACKAGE);
        } else {
                write("I -> Can't send command package. Bluetooth buffer is full ");
        }

 }

 private synchronized void releaseLock(){
        notifyAll();
 }

/************************Output to the application ***********************************/

 /**
  * Write to API
  *
  * @param String Text that will be written
  */
 public void write(String s){
        application.stackInformation(s + "at " +myPort);
 }

 /**
```

```
 * Write to API
 * String + byte vector
 *
 * @param String Explanation
 * @param byte[] byte vector
 */
public void writeByte(String s, byte[] b){

        String temp="";;

        temp=temp+s;
         for (int i=0;i<b.length;i++){
                 temp = temp + String.valueOf(b[i])+ " ";
         }

          temp=temp+"at " +myPort;
          application.stackInformation(temp);
}

 /**
 * Write to API
 * String + byte vector. Length number of byte is written.
 *
 * @param String Explanation
 * @param byte[] byte vector
 * @param int Length of byte vector must be smaller or equal with length of byte vector
 */
public void writeByte(String s, byte[] b,int length){

        String temp="";

        if (length < b.length){
                temp=temp+s;
                for (int i=0;i<length;i++){
                        temp = temp + String.valueOf(b[i])+ " ";
                }

                temp=temp +"at " +myPort;
                application.stackInformation(temp);
        }

}


/** Write to API in form the String + byte vector.
* Start writing at position startPos in byte vector.
*
* @param byte[] byte vector
* @param int start position in byte vector
* @param String Explanation
*/
public void writeByte(byte[] b, int startPos, String s){

        String temp="";
        temp=temp+s;

         for (int i=startPos;i<b.length;i++){
                 temp = temp + String.valueOf(b[i])+ " ";
        }
```

```
		temp=temp +"at " +myPort;
		application.stackInformation(temp);
	}


} // end class ControllLayer

/********************************* Help classes*************************************/
/**
 * The class contains a accessible list of all Bluetooth devices.
 */
class HostList{

		private Vector myVector;

		/**
		* Constructor
		*/
		public HostList(){
			myVector = new Vector();
		}

		/**
		 *  Return a stored host object
		 *
		 * @param byte[] The address or CID of the host
		 * @return The found Host object otherwise null
		 */
		public synchronized Host getHost(byte[] adress){

			Host temp;

			for (int i=0;i<myVector.size();i++){
				temp = (Host) myVector.elementAt(i);
				if(BlackBox.equals(adress,temp.getAdress())){
					return temp;
				}
			}

			for (int i=0;i<myVector.size();i++){
				temp = (Host) myVector.elementAt(i);
				if(BlackBox.equals(adress,temp.getCID())){
					return temp;
				}
			}

			return null;
		}

		/**
		 * Return host at index
		 *
		 * @param int index
		 * @param A stored Host
		 */
		 public synchronized Host getHostAt(int index){

			if(index < 0 || index >= myVector.size()){
				return null;
			}
```

```
                return (Host) myVector.elementAt(index);
        }

        /**
         * Store Host object in class
         *
         * @param Host The object
         */
        public synchronized void storeHost(Host s){
                myVector.addElement(s);
        }

        /**
         * Return number of stored hosts
         *
         * @param Stored hosts
         */
        public synchronized int getSize(){
                return myVector.size();
        }



        /**
         * Remove host from class
         *
         * @param The host that will be removed
         */
        public synchronized void removeHost(Host s){
                myVector.remove(s);
        }
}

/**
 * A Bluettooth device
 */
class Host{

        private byte[] BD_Adress;
        private byte[] CID;
        private byte[] connectionHandle;
        private byte pageScanRepetition;
        private byte pageScanPeriod;
        private byte pageScanMode;
        private byte[] clockOffset;
        private boolean isConnected;
         private boolean status;


        /*
        * Constructor
        *
         * @param byte[] The Bluetooth adress
        * @param byte scanRep page scan repetition of the connection
        * @param byte scanPeriod page scan period mode of the connection
         * @param byte scanMode page scan mode of the connection
         * @param byte[] offset, the clock offset of the connection
         */
        public Host(byte[] adress, byte scanRep, byte scanPeriod,
```

```
                                byte scanMode, byte[] offset){

                BD_Adress= new byte[6];
                 CID = new byte[2];
                 clockOffset = new byte[2];
                 connectionHandle = new byte[2];

                pageScanRepetition = scanRep;
                pageScanPeriod = scanPeriod;
                 pageScanMode = scanMode;

                for(int i=0;i<adress.length;i++){
                        BD_Adress[i]=adress[i];
                  }

                for(int i=0;i<offset.length;i++){
                        clockOffset[i]=offset[i];
                  }

                 connectionHandle[0]=0;
                 connectionHandle[1]=0;

                 CID[0]=0;
                 CID[1]=1;

                 isConnected = false;
                status = false;

}


/****************************Get Operation ******************************/
public byte[] getAdress(){
        return BD_Adress;
}

public byte[] getCID(){
        return CID;
}

public byte getScanRepetition(){
        return pageScanRepetition;
}

public byte getScanPeriod(){
        return pageScanPeriod;
}

public byte getScanMode(){
        return pageScanMode;
}

public byte[] getConnectionHandle(){
        return connectionHandle;
}

public byte[] getClockOffset(){
        return clockOffset;
}
```

```
	public boolean getConnected(){
		return isConnected;
	}

	public boolean getStatus(){
		return status;
	}

	/*******************************Set Operation ********************************/
	public void setConnectionHandle(byte[] b){
		connectionHandle[0]=b[0];
		connectionHandle[1]=b[1];
	}

	public void setCID(byte[] b){
		CID[0] = b[0];
		CID[1] = b[1];
	}

	public void setConnected(boolean connect){
		isConnected = connect;
	}

	public void setStatus(boolean s){
		status=s;
	}
}
```

---------- L2CAP ----------

```
/**
 * L2CAP layer
 */
class L2CAP{

        private int numOfPackage;
        private int sizeOfPackage;
        private ControllLayer app;
        private HCI myHCILayer;
        private boolean debug;

  /**
   * Constructor
   *
   * @param int num Maximal number of packages in Bluetooth buffer
   * @params int size maximal size of data package in Bluetooth buffer
   * @param ControllLayer controls the stack
   * @param HCI the HCI layer
   */
  public L2CAP(int num,int size,ControllLayer theController,HCI theHCILayer){

         numOfPackage = num;
        sizeOfPackage = size;
         app = theController;
         myHCILayer = theHCILayer;
        debug = true;
  }

 /**
   * Flow Control for ACL/data package
   *
   * @param int Maximal Bluetooth ACL buffer
   * @param int Maximal size of a ACL package
   */
  public void setFlowControl(int num,int size){

         numOfPackage = num;
          sizeOfPackage = size;

        if (debug) {
                app.write("I -> FlowControl, ACLBuffer = " +String.valueOf(num) +
                        " ACLPackageSize = " + String.valueOf(size) + " ");
        }
  }

  /**
   * Acknowledge from Bluetooth host. The host has sent send number of packages
   * and the buffer increases with that size
   *
   * @param int Number of sent package from Bluetooth buffer
   */
  public void finishedSendPackage(int bufferIncrease) {

                numOfPackage += bufferIncrease;

                if (debug){
                        // app.write(" I -> Bluetooth buffer size " + String.valueOf(numOfPackage) +
                                " packages ");
```

```
                }
    }

    /**
     * Send ACL package.
     *
     * @param byte[] The CID of the connection
     * @param byte[] The connectionHandle
     * @param byte[] the data package. All byte allocated and the data is inserted
     * @return true if sucess otherwise false
     */
    public boolean sendACL(byte[] CID,byte[]connectionHandle,byte[] data){

                byte[] packageHCILength;
                byte[] packageL2CAPLength;

                //A fix. It can otherwise in strange be set to zero
                numOfPackage=10;

                // Package size less then maximal size
                if (((int) (data.length + 9) <= sizeOfPackage) && (numOfPackage >= 1)){

                        packageL2CAPLength=new byte[2];

                        // length of L2CAP head
                        packageL2CAPLength[0] = (byte) ((short) (data.length-9));
                        packageL2CAPLength[1] = (byte) ((short) ((data.length-9)>>8));


                        data[5] = packageL2CAPLength[0];
                        data[6] = packageL2CAPLength[1];
                        data[7] = CID[0];
                        data[8] = CID[1];

                        myHCILayer.sendData(data,BlackBox.ACLPACKAGE,connectionHandle);
                        numOfPackage--;
                        return true;

                // flow control for data packages
                } else {
                        app.write("E -> L2CAP Bluetooth buffer full ");
                        return false;
                }

    }

    /**
     * Receive package from Bluetooth
     *
     * @param byte[] InPackage
     * @param int the position of the first byte in the vector
     */
    public void receiveACL(byte[] data,int pos){

            //peel of L2CAP package head
            app.receiveACL(data,pos+4);

    }

} // end class L2CAP
```

---------- Hci ----------

```
/**
 *HCI layer in the Bluetooth stack
 */
class HCI{

        private RS232 myTransmitt;
         private L2CAP myL2CAP;
        private ControllLayer app;

 /**
  * Constructor
  *
  * @param The controll layer
  */
 public HCI(ControllLayer blue){

  app = blue;

 }

 /**
  * Register L2CapLayer
  *
  * @param L2Cap My L2Caplayer
  * @param RS232 The serial port
  */
 public void setInitValue(L2CAP cap,RS232 r){

         myL2CAP = cap;
         myTransmitt = r;

 }

 /**
 * Receiving ACL or EventPacket from RS232 layer. Always all data in the package
 *
 * @param byte[] One package from serial port
 * @param int start position of the first byte
 */
 public void reciveData(byte[] data,int pos){

        byte[] temp;
        byte[] storedData;

         // data package. The packet has always all of it's byte in one package
        if (data[pos] == 0x02){

                // must contain data and the conection handler must be one. Error check that can be removed
                if( data.length > 9) {

                        if(data[pos+1] == 0x01){
                                //peel of package type and HCI head
                                myL2CAP.receiveACL(data,5);
                        } else {
                                // This message has never been writen
                                app.write("E -> Error synchronization incomming package ");
                        }
```

```
                } else {
                        app.write("E -> Incomming data package has no data ");
                 }

            // EventPackage
            } else if(data[pos] == 0x04){

                //peel of package type
                app.receiveEvent(data,1);

            } else {

                //Wrong input package
                 app.write("E -> Error input package type. ");

           }

}

/**
* Sending a command package to serial port
*
* @param String The Data that will be sent
* @param int Type of package se BlackBox
*/
public void sendData(byte[] data, int type){

        if( type == BlackBox.COMMANDPACKAGE){

                String inData="";

                data[0]=0x01;
                for(int i=0;i<data.length;i++){
                        inData=inData + "0x" + UnicodeFormatter.byteToHex(data[i]) + " ";
                }

                app.write("I -> Sending command " + inData);
                myTransmitt.sendData(data);

        } else {

                app.write(" E -> Trying to send datapackage as commandpackage ");
        }

}

/**
* Sending a data packet to serial port. The data package from L2CAP
* does always fit in one hci package so there is no need for segmentation-
*
* @param String The Data that will be sent
* @param int Type of package se BlackBox
* @param byte[] The connectionhandle
*/
public void sendData(byte[] data, int type,byte[] connectionHandle){

        byte[]packageHCILength;

        packageHCILength = new byte[2];
```

```
        if( type == BlackBox.ACLPACKAGE){

                data[0]=0x02;

                // put in BC and PB flag of HCI head

                connectionHandle[1]= (byte) (connectionHandle[1] & 0x0f);
                connectionHandle[1] += 0x20;

                // length of HCI Head
                packageHCILength[0] = (byte) ((short) (data.length-5));
                packageHCILength[1] = (byte) ((short) ((data.length-5)>>8));

                data[1]=connectionHandle[0];
                data[2]=connectionHandle[1];
                data[3]=packageHCILength[0];
                data[4]=packageHCILength[1];

                myTransmitt.sendData(data);

         } else {
                app.write(" E -> Trying to send commandpackage as data package ");
        }

}

                                                        }
```

---------- RS232 ----------

```
import java.io.*;
import java.util.*;
import javax.comm.*;

/**
* This class listens from serial port and buffer package. When the class has received a package it is sent to
* the HCI layer. It also transmit packages over the serial port
 */
class RS232 implements SerialPortEventListener {


        private CommPortIdentifier portId;
        private Enumeration portList;
        private SerialPort serialPort;
        private OutputStream outputStream;
        private InputStream inputStream;

        // For debug
        public int variabel;

        private ControllLayer app;
        private boolean newPackage;
        private boolean readLength;
        private int packageType;
        private int nbrOfBytes;
        private int packageLength;
        private byte packet[];
        private byte temp_vector[];
        private int pos;
        private HCI myHCILayer;
        private boolean debug;
        private String myPort;


        /**
         * Constructor
         *
         * @param String COM port to connect
        *  @param HCI HCILayer
        *  @param ControllLayer theApp for debug information
         */
         public RS232(String connectPort,HCI theHCILayer,ControllLayer theApp) {

                packet=new byte[BlackBox.MAXSIZE];
                packageType = BlackBox.NOPACKAGE;
                newPackage=true;
                readLength=false;
                debug=true;
                pos=0;
                myPort=connectPort;
                app=theApp;

                myHCILayer = theHCILayer;

                  portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
                  portId = (CommPortIdentifier) portList.nextElement();
```

```
                    if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                    if (portId.getName().equals(connectPort)) {

                            try {
                            serialPort = (SerialPort) portId.open("Receive", 2000);
                            } catch (PortInUseException e) {
                                    app.write("PortInUseException " + e.toString()+ " ");
                            }
                            try {
                            inputStream = serialPort.getInputStream();
                            outputStream = serialPort.getOutputStream();
                            } catch (IOException e) {
                                    app.write("Error creating stream " + e.toString()+ " ");
                            }
                            try {
                                    serialPort.addEventListener(this);
                            } catch (TooManyListenersException e) {
                                    app.write("TooManyListenerException "+ e.toString()+ " ");
                            }

                            serialPort.notifyOnDataAvailable(true);
                             try {
                                    serialPort.setSerialPortParams(57600,
                                    SerialPort.DATABITS_8,
                                    SerialPort.STOPBITS_1,
                                    SerialPort.PARITY_NONE);
                            } catch (UnsupportedCommOperationException e) {
                                    app.write("CommOperationException at port " + e.toString()+ " ");
                            }

                    }
                    }
            }

             if(inputStream != null){
                    //OK
             } else {
                    app.write("E -> Couldn't create RS-232 receiver ");
            }
    }

/**
* Send data over serial port
*
* @param byte[] The data
*/
public synchronized void sendData(byte[] b){

            /*
            if (debug){
                    app.writeByte("I -> Sending byte vector: ",b);
            }
            */

             try {

                    outputStream.write(b);

            } catch (IOException e) {
                    app.write("E -> IOException writing data " + e.toString()+ " ");
```

```
        } catch (Exception e){
                        app.write("E -> Generall exception transmitting " + e.toString() + " ");
        }

}

   /**
    * Listen on event from serial port
    *
    * @param SerialPortEvent
    */
   public synchronized void serialEvent(SerialPortEvent event) {

      switch(event.getEventType()) {
      case SerialPortEvent.BI: app.write("E -> SerialPortEvent.BI ");
                                                      break;
      case SerialPortEvent.OE: app.write("E -> SerialPortEvent.OE ");
                                                      break;
      case SerialPortEvent.FE: app.write("E -> SerialPortEvent.FE ");
                                                      break;
      case SerialPortEvent.PE: app.write("E -> SerialPortEvent.PE ");
                                                      break;
      case SerialPortEvent.CD: app.write("E -> SerialPortEvent.CE ");
                                                      break;
      case SerialPortEvent.CTS: app.write("E -> SerialPortEvent.CTS ");
                                                      break;
      case SerialPortEvent.DSR: app.write("E -> SerialPortEvent.DSR ");
                                                      break;
      case SerialPortEvent.RI: app.write("E -> SerialPortEvent.RI ");
                                                      break;
      case SerialPortEvent.OUTPUT_BUFFER_EMPTY: app.write("E ->
                                              SerialPortEvent.OUTPUT_BUFFER_EMPTY ");
                                                      break;
      case SerialPortEvent.DATA_AVAILABLE:

          int numBytes;
          int temp;

          numBytes=0;

          try {
                  byte[] rb = new byte[20];

                  while (inputStream.available() > 0) {
                          numBytes = inputStream.read(rb);
                  }

                  //app.writeByte("I -> SerialPortEvent.DATA_AVAILABLE ",rb,numBytes);

                  //Store input byte
                  for (int i=0;i<numBytes;i++){

                          if(pos == 0 && rb[i] == 0){
                                  // Do not read if zero garbage bytes
                          } else {
                                  packet[pos]=rb[i];
                                  pos++;

                                  //Do not read other garbage byte at front
                                  if( pos == 1 && ( rb[0] != 0x02 && rb[0] != 0x04) ){
```

```
                        pos--;
                    }
                }
            }

            do {

            // What type of package is it
            if (pos > 0 && newPackage) {
                newPackage = false;
                if (packet[0] == 0x02) {            //DataPackage
                    packageType = BlackBox.ACLPACKAGE;
                } else if (packet[0] == 0x04) { // EVENTPACKAGE
                    packageType = BlackBox.EVENTPACKAGE;
                } else{

                    //Garbage byte
                    //app.write("E -> Wrong packettype at " + myPort
                    //      +". Package start with byte "
                    //      + String.valueOf(packet[0])
                    //      );

                    //Garbage bytes between incoming package. Remove the byte
                    int k =0;
                    while(k<pos && (packet[k] != 0x02 && packet[k] != 0x04)){
                        k++;
                    }

                    pos=pos-k;
                    for(int t=0; t<pos;t++){
                        packet[t]=packet[t+k];
                    }

                    newPackage=true;
                    packageType = BlackBox.NOPACKAGE;

                }
            }


            // how long is data package
            if(packageType ==BlackBox.ACLPACKAGE && pos >=4 && readLength ==false){
                readLength=true;
                packageLength = (int) packet[3];
                temp=(int) packet[4];
                temp=temp<<8;
                packageLength =packageLength | temp;
            }


            // how long is event package
            if(packageType ==BlackBox.EVENTPACKAGE && pos >=2 && readLength ==false){
                readLength=true;
                packageLength = (int) packet[2];
            }
```

```
// have I read the whole data package
if(packageType == BlackBox.ACLPACKAGE && pos >= (packageLength + 5) &&
readLength == true){
        readLength=false;
        newPackage=true;

        // received bytes from another package
        if (pos > (packageLength +5)){

                temp_vector = new byte[packageLength+5];
                for (int k=0;k<(packageLength+5);k++){
                        temp_vector[k]=packet[k];
                }

                for (int k=(packageLength+5);k<pos;k++){
                        packet[k-(packageLength+5)]=packet[k];
                }

                pos=pos-(packageLength+5);
                PackageJob job=new PackageJob
                                (this,myHCILayer,temp_vector,temp_vector.length,app);
                job.start();

        } else {
                pos=0;

                PackageJob job=new PackageJob
                                (this,myHCILayer,packet,packageLength+5,app);
                job.start();
        }

         packageType=BlackBox.NOPACKAGE;

}


// Have I read the whole event package
if(packageType == BlackBox.EVENTPACKAGE && pos >= (packageLength + 3) &&
readLength == true){
        readLength=false;
        newPackage=true;

        // Received bytes from another package
        if (pos > (packageLength +3)){

                temp_vector = new byte[packageLength+3];
                for (int k=0;k<(packageLength+3);k++){
                        temp_vector[k]=packet[k];
                }

                for (int k=(packageLength+3);k<pos;k++){
                        packet[k-(packageLength+3)]=packet[k];
                }

                pos=pos-(packageLength+3);

                PackageJob job=new PackageJob
                        (this,myHCILayer,temp_vector,temp_vector.length,app);
                job.start();
```

```
                        } else {

                                pos=0;

                                PackageJob job=new PackageJob
                                        (this,myHCILayer,packet,packageLength+3,app);
                                job.start();

                        }

                         packageType=BlackBox.NOPACKAGE;

                }

        // the data buffer still contains a whole package
        } while(pos > 19);

        } catch (Exception e) {
                app.write(" E -> Exception while recieving data ");
                app.write(e.toString());
        }

       break;

      default      : app.write("E -> Error strange serial event ");
    }
  }
} // end RS232

/************************The thread class that contains a full package ********************/
/**
 * Package that are recived from the serial port
 */
class PackageJob extends Thread{

  private HCI myHCI;
  private ControllLayer app;
  private RS232 t;
  private byte[] data;

 /**
  * Constructor
  *
  * @param Rs232 test. To detect number of threads in the system
  * @param HCI the HCI layer
  * @param String The data that will be transmitted
  * @param int theLength number of bytes
  * @param ControllLayer theApp for debug information
  */
 public PackageJob(RS232 test,HCI theHCI,byte[] theData,int theLength,ControllLayer theApp){

        super();

        myHCI=theHCI;
        t=test;
        app=theApp;

        data = new byte[theLength];
```

```
            //Copy the data/event package
            for(int i=0;i<theLength;i++){
                        data[i]=theData[i];
            }

            //Increase number of thead in system
            t.variabel++;
  }

  /**
   * Run method
   */
  public void run(){

            myHCI.reciveData(data,0);

            // decrease number of thread in system
             t.variabel--;
            // debug information
             if(t.variabel != 0 && t.variabel%100 == 0){
                        app.write("I -> Number of in pacakge threads in system: " + String.valueOf(t.variabel));
             }
  }

} // end PackageJob
```

---------- Help classes in the stack ----------

```
/*
* The class contains definition and global variable and methods used in the stack. It provide a way of changing
variable all
* over the stack without changing the specific files.
*/
class BlackBox{

  public static int NOPACKAGE=-1;
  public static int COMMANDPACKAGE=0;
  public static int ACLPACKAGE=1;
  public static int EVENTPACKAGE=2;

  public static int TRANSMITTPACKAGE = 3;

  public static int FORWARD=0;
  public static int LEFT=1;
  public static int RIGHT=2;
  public static int REVERSE=3;

  // maximum size of payload transfered between bluetoth units, not used ?
  public static int MAXSIZE = 10000;
  public static int MAXSIZE_PACKAGE = 800;

  public static double MAX_ANALOG = 5.0;                    // max voltages
  public static double MAX_PERIOD = 4;                      // max period length
  public static double MAX_SAMPLE_TIME = 255;               // max sample time in BCC


  /**
   * Return true if vector one has the same order and length of bytes as vector two.
   *
   * @param byte[] First vector
   * @param byte[] Second vector
   * @return True if First vector = second vector otherwise false
   */
  public static boolean equals(byte[] a, byte[] b){

          if(a.length != b.length){
                  return false;
          }

          for (int i = 0; i<a.length;i++){

                  if(a[i] !=b[i]){
                          return false;
                  }

          }

          return true;
  }

  /**
 * Convert a vector  of byte to a string and
* switch the order of the byte byte[0] MSB
*
* @param byte[] The byte vector with the length of 6 byte
*/
```

```
public static String convertToString(byte[] hostAdress){

        String temp;

         temp = String.valueOf(hostAdress[5])+String.valueOf(hostAdress[4])+
                String.valueOf(hostAdress[3])+String.valueOf(hostAdress[2])+
                        String.valueOf(hostAdress[1])+String.valueOf(hostAdress[0]);

                return temp;
        }

}


/*
 * Copyright (c) 1995-1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies. Please refer to the file "copyright.html"
 * for further important copyright and licensing information.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */

import java.io.*;

class UnicodeFormatter  {

  static public String byteToHex(byte b) {
    // Returns hex String representation of byte b
    char hexDigit[] = {
      '0', '1', '2', '3', '4', '5', '6', '7',
      '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
    };
    char[] array = { hexDigit[(b >> 4) & 0x0f], hexDigit[b & 0x0f] };
    return new String(array);
  }

  static public String charToHex(char c) {
    // Returns hex String representation of char c
    byte hi = (byte) (c >>> 8);
    byte lo = (byte) (c & 0xff);
    return byteToHex(hi) + byteToHex(lo);
  }

} //class
```

Appendix E: Code in Bluetooth Controlled Beetle

----------CarPort----------

```
import java.util.Vector;
import java.awt.*;
import javax.swing.*;

public class CarPort implements API{

	private Car myCar;
	private ControllLayer comStack1;
	private byte[] address,cid;

	/**
	* Constructor
	*/
	public CarPort(Car blue, byte[] radiocarAddress){
		myCar=blue;
		address=radiocarAddress;
	}

	/*
	* Store The channal of the connection
	*/
	public void storeCID(byte[] temp){

		cid=new byte[2];
		cid[0]=temp[0];
		cid[1]=temp[1];
	}

	/***************************** Stack to application ******************************/
	/*
	* Receiving data from bluetooth stack
	*
	* @param byte[] data received
	* @param pos start position of the first byte
	*/
	public void receiveData(byte[] data, int pos){
									//do nothing
	}

	/*
	* Error event from bluetooth received
	*
	* @param byte[] event
	*/
	public void receiveEvent(byte[] event){
		String s=new String();
		for(int k=0;k<event.length;k++){
			s = s +" " + UnicodeFormatter.byteToHex(event[k]);
		}
		myCar.write("E -> Strange event at radiocar: " + s);
	}

	/*
	* Bluetooth unit discovered
	*
	* @param byte[] The address of the bluetooth unit
```

```
*/
public void BTFound(byte[] address){

        //Don't update GUI if the stack finds the joystick
        if(!BlackBox.equals(myCar.joyAddress,address)){
                myCar.newHost(address);
                myCar.write("I--> New host at CarPort");
        } else {
                        //joystick
        }
}

/*
* Close connection from other Bluetooth unit
*/
public void connectionIsClosed(byte[] address){
        byte[] temp;

        myCar.setCarConn(false);
        myCar.myGUI.setRightButtonMode();
        myCar.myGUI.connectionClosed(address);
        myCar.write("I --> Connection is closed at CarPort");
        temp = comStack1.createConnection(address);
        while(temp[0]==0x00&&temp[1]==0x00){
                myCar.write("E --> New connection to car failed");
                temp=comStack1.createConnection(address);
        }

        myCar.myGUI.writeConnected(address);
        myCar.setCarConn(true);
        myCar.myGUI.setRightButtonMode();
}

/*
* Information if stack exucuation
*
* @param String The information
*/
public void stackInformation(String t){
        myCar.write(t + " at CarPort");
}

/*
* Information about the RSSI value
*/
public void receiveRSSI(byte value){
       //set something in myCar
         //System.out.println("RSSI 0x" + UnicodeFormatter.byteToHex(value));
}
```

```
/************************** Application to stack ***************************************/
	/**
	* Methods that the application can call in the bluetooth stack
	*
	* boolean sendData(byte[] CID,byte[] data);
	* boolean init();
	* boolean inquiry();
	* byte[] createConnection(byte[] address);
	* boolean closeConnection(byte[] btAddress);
	* boolean readRSSI(byte[] btAddress);
	* reset(byte[] btAdress)
	*/

	/*
	* init comPort at computer
	* return true if success otherwise false
	*/
	public boolean initComPort(int comPort){

		if(comStack1 == null){
			comStack1 = new ControllLayer("COM1",this);
			return true;
		} else {
			return false;
		}
	}

	/*
	* init bluetooth card at computer
	*/
	public void initBT(){

		CarPackageJob temp = new CarPackageJob(myCar.myGUI,myCar,comStack1,1);
		temp.start();
	}

	/*
	* Inquiry for bluetooth host
	*/
	public void inquiry(){

		CarPackageJob temp = new CarPackageJob(myCar.myGUI,myCar,comStack1,2);
		temp.start();
	}

	/*
	* Create connection with car
	* @param byte[] bluetooth address
	*/
	public void createConnection(byte[] address){

		CarPackageJob temp = new CarPackageJob(myCar.myGUI,myCar,comStack1,3);
		temp.storeData(address);
		temp.start();
	}

	/*
	* Send data to car
	*
	* @param byte[] The channel
```

```
		* @param byte[] The data
		*/
		public void sendData(byte[] data){
			comStack1.sendData(cid,data);
		}

		/*
		* Close connection
		*
		* @param byte[] The BT-address
		*/
		public void closeConnection(byte[] address){
			myCar.setCarConn(false);
			myCar.myGUI.setRightButtonMode();
			comStack1.closeConnection(address);
		}

		/*
		* Software reset on bluetooth stack
		*/
		public void reset(){
			myCar.setCarConn(false);
			myCar.myGUI.setRightButtonMode();
			comStack1.reset();
		}

		public void readRSSI(){
			comStack1.readRSSI(address);
		}

}

/**
 * Package that will be transmitted over the RS-232 to bluetooth device
 */
class CarPackageJob extends Thread{

  private GUI myGUI;
  private Car myCar;
  private ControllLayer myStack;
  private byte[] dataByte;
  private int command;

 /**
  * Constructor
  *
  * @param GUI the grapichal interface
  * @param int command 1 = init
  *                    2 = inquiry
  *                    3 = createConnction
  */
 public CarPackageJob(GUI theGUI, Car theCar,ControllLayer theStack,int theCommand){

		super();

		myGUI = theGUI;
		myCar=theCar;
		myStack=theStack;
		command=theCommand;
```

```
		//in case
		dataByte = new byte[1];
	}

	/**
	 * Run method
	 */
	public void run(){

		byte[] temp;
		boolean sucess;

		switch(command){

			//Init
			case 1 :  myCar.write("I -> Init BTcard at carPort...");
				  sucess=myStack.init();
				  myGUI.initCOM1Done(sucess);
				  myCar.setCarInitialized(true);
				  break;

			//inquiry
			case 2 :  myCar.write("I -> Inquiry at carPort....");
				  sucess=myStack.inquiry();

				  //write error mesages
				  if(sucess){
					myCar.write("I--> Inquiry sucess at CarPort...");
				  } else{
					myCar.write("E--> Inquiry failed at CarPort...");
				  }
				  myGUI.inquiryDone();
				  break;

			//CreateConnection
			case 3 :  myCar.write("I -> Connecting to Car...");
				  temp=myStack.createConnection(dataByte);

				  //update GUI
				  if(temp[0]==0x00 && temp[1]==0x00){
					myCar.write("E --> New connection to car failed");
					myGUI.connectionFailed(dataByte);
				  } else {
					myGUI.connectionSuccess(dataByte);
					myCar.myCarPort.storeCID(temp);
					myCar.setCarConn(true);

					if(myCar.getControlMode()== myGUI.stopMode){
						myGUI.setRightButtonMode();
					}else{
						myGUI.setStopButtonMode();
					}

				  }
				  break;

			//Error
			default:  myCar.write("E -> Trying to make illegal operation to controllLayer");
				  break;
		}
```

```
  }

  /*
  * The data of the package are stored
  *
  * @param byte[] the data
  */
  public void storeData(byte[] data){

        dataByte = new byte[data.length];

        for(int i=0;i<data.length;i++){

              dataByte[i]=data[i];
        }

  }


}
```

----------JoyPort----------

```
import java.util.Vector;
import java.awt.*;
import javax.swing.*;

public class JoyPort implements API{

	private Car myCar;
	private ControllLayer comStack2;
	private byte[] address,cid;
	private int count=0;
	private double  maxSpeed=1;
	private boolean minSpeed=false;

	public byte btemp=0x45;
	public byte ttemp=0x40;
	public boolean init;

	/**
	* Constructor
	*/
	public JoyPort(Car blue, byte[] joyAddress){
		myCar=blue;
		address=joyAddress;
		init=false;
	}

	/*
	* Store The channal of the connection
	*/
	public void storeCID(byte[] temp){

		cid=new byte[2];
		cid[0]=temp[0];
		cid[1]=temp[1];
	}
	/***************************** Stack to application **********************/

	/*
	* Receiving data from bluetooth stack
	*
	* @param byte[] data received
	* @param pos start position of the first byte
	*/
	public void receiveData(byte[] data, int pos){

		double d;
		byte b,b1;
		byte speed=0x00;
		byte steering=0x00 ;
		byte check=0x00;

		//conver the analog input one by removing 2 LSB
		check = (byte)(data[pos+1]<<6);
		b1 = (byte)((data[pos] & 0xff) >>> 2);
		check = (byte) (check | b1);
		d = convertToDouble(check);

		// make sure that outputvoltage is correct at the beginning. Over 1 volt
		if(!init && d <= 51){
```

```
                myCar.write("E --> Init voltage to low at joystick resending start package");

                // resend first data package
                byte[] joyData=new byte[11];
                joyData[0]=(byte)0xff;      //analogout1=5V
                joyData[1]=(byte)0x03;
                joyData[2]=(byte)0xff;      //analogout2=5V
                joyData[3]=(byte)0x03;
                joyData[4]=(byte)0xfe;      //sets PWM period to 4.2 ms
                joyData[5]=0x00;            //sets the PWM dutycycles to zero
                joyData[6]=0x00;
                joyData[7]=0x00;
                joyData[8]=0x00;
                joyData[9]=0x14;            //sets sampling sampling time=20ms
                joyData[10]=0x00;                   //sets SPI off
                sendData(joyData);

        } else {

                init=true;

                b1=0x00;
                b=0x00;

                speed = (byte) (data[pos+3]<<6);
                b1 = (byte) ((data[pos+2] & 0xff) >>> 2);
                speed = (byte) (speed | b1);

                steering = (byte)(data[pos+1]<<6);
                b1 = (byte)((data[pos] & 0xff) >>> 2);
                steering = (byte) (steering | b1);

                d = convertToDouble(steering);
                b = convertToHex(((65*d)/255)+115);
                myCar.setSteering(b);

                if(minSpeed){
                        d = convertToDouble(speed);
                        d = ((33.0*d)/255.0);
                        d = (d-13.5) +13.5;
                        d = d+60.5;
                        if(d>77){
                                d=77;
                        }else if(d<71){
                                d=71;
                        }
                        b = convertToHex(d);
                }else{
                        d = convertToDouble(speed);
                        d = ((33.0*d)/255.0);
                        d = (d-13.5)*maxSpeed +13.5;
                        d = d+60.5;
                        b = convertToHex(d);
                }
                if(b==0x4c){
                        b=0x00;
                }else if(b==0x4b){
                        b=0x00;
                }else if(b==0x4a){
```

```
                        b=0x00;
                }else if(b==0x49){
                        b=0x00;
                }

                myCar.setSpeed(b);

        }

}

/*
* Error event from bluetooth received
*
* @param byte[] event
*/
public void receiveEvent(byte[] event){
        String s= new String();
        for(int k=0;k<event.length;k++){
                s = s + " "+ UnicodeFormatter.byteToHex(event[k]);
        }
        myCar.write("E -> Strange event at joystick: " + s);
}

/*
* Bluetooth unit discovered
*
* @param byte[] The address of the bluetooth unit
*/
public void BTFound(byte[] address){

        //Don't update GUI if the stack finds the car
        if(!BlackBox.equals(myCar.carAddress,address)){
                myCar.newHost(address);
                myCar.write("I--> New host at JoystickPort");
        } else {
                //The car
        }
}

/*
* Close connection from other Bluetooth unit
*/
public void connectionIsClosed(byte[] address){
        byte[] temp;

        //stop the car
        myCar.setControlMode(2);

        myCar.setJoyConn(false);
        myCar.myGUI.setRightButtonMode();
        myCar.myGUI.connectionClosed(address);
        myCar.write("I --> Connection is closed at JoystickPort");
        temp = comStack2.createConnection(address);
        while(temp[0]==0x00 && temp[1]==0x00){
                myCar.write("E --> New connection to Joystick failed");
                temp = comStack2.createConnection(address);
        }

        init=false;
```

```
            byte[] joyData=new byte[11];
            joyData[0]=(byte)0xff;      //analogout1=5V
            joyData[1]=(byte)0x03;
            joyData[2]=(byte)0xff;      //analogout2=5V
            joyData[3]=(byte)0x03;
            joyData[4]=(byte)0xfe;      //sets PWM period to 4.2 ms
            joyData[5]=0x00;            //sets the PWM dutycycles to zero
            joyData[6]=0x00;
            joyData[7]=0x00;
            joyData[8]=0x00;
            joyData[9]=0x14;            //sets sampling sampling time=20ms
            joyData[10]=0x00;                   //sets SPI off
            sendData(joyData);

            myCar.myGUI.writeConnected(address);
            myCar.setJoyConn(true);
            myCar.myGUI.setRightButtonMode();
}

/*
* Information if stack exucuation
*
* @param String The information
*/
public void stackInformation(String t){
            myCar.write(t+" at JoystickPort");
}

/*
* Information about the RSSI value
*/
public void receiveRSSI(byte value){
        //do nothing
}

/************************** application to stack ***********************************/
/**
* Methods that the application can call in the bluetooth stack.
*
* boolean sendData(byte[] CID,byte[] data);
* boolean init();
* boolean inquiry();
* byte[] createConnection(byte[] address);
* boolean closeConnection(byte[] btAddress);
* boolean readRSSI(byte[] btAddress);
*/

/*
* init bluetooth card at computer
*
* return true if success otherwise false
*/
public boolean initComPort(int comPort){
                    if (comStack2 == null){
                                comStack2 = new ControllLayer("COM2",this);
                                return true;
                    } else {
                                return false;
                    }
}
```

```
/*
* init bluetooth card at computer
*/
public void initBT(){

	JoyPackageJob temp = new JoyPackageJob(myCar.myGUI,myCar,comStack2,1);
	temp.start();
}

/*
* Inquiry for bluetooth host
*/
public void inquiry(){

	JoyPackageJob temp = new JoyPackageJob(myCar.myGUI,myCar,comStack2,2);
	temp.start();

}


/*
* Create connection with joystick
*
* @param byte[] bluetooth address
*/
public void createConnection(byte[] address){

	JoyPackageJob temp = new JoyPackageJob(myCar.myGUI,myCar,comStack2,3);
	temp.storeData(address);
	temp.start();

}

/*
* Send data to joystick
*
* @param byte[] The data
*/
public void sendData(byte[] data){
	comStack2.sendData(cid,data);
}

/*
* Close connection
*
* @param byte[] The BT-address
*/
public void closeConnection(byte[] address){

	myCar.setJoyConn(false);
	myCar.myGUI.setRightButtonMode();
	comStack2.closeConnection(address);
}


/*
* Software reset on bluetooth stack
*/
public void reset(){
```

```
                myCar.setJoyConn(false);
                myCar.myGUI.setRightButtonMode();
                comStack2.reset();
        }
        /************************* other methods *********************************/

/**
  * Converts a double between 0-255 to hex
  *
  * @param double The analog value that will be converted
  */
 private byte convertToHex(double d){

        double temp;
        byte b = 0x00;
        byte mask = (byte) 0x80;

        for (int i=7;i>=0;i--){

                if(d/((double) Math.pow(2,i)) >= 1){

                        d -= Math.pow(2,i);
                        b = (byte) (b | mask);
                }
                        mask = (byte) ((mask & 0xff) >>> 1);
        }

        return b;

 }

 /**
  * Converts the byte to a double
  *
  * @param the byte containing the data max
  */
 private double convertToDouble(byte b){

        double sum =0;
        byte mask = 0x01;

        for (int i=0;i<8;i++){

                if (((b & 0xff) & mask) != 0x00){
                        sum = sum+ Math.pow(2,i);
                }

                mask = ((byte) (mask<<1));
        }

        return sum;

 }

public synchronized void setMaxSpeed(int sp){
        maxSpeed=(double)sp/30;
        if(sp==6){
                minSpeed=true;
        }else{
                minSpeed=false;
```

```
		}
}

public synchronized void addOne(){
		btemp++;
		ttemp++;

		System.out.print("steering"+ UnicodeFormatter.byteToHex(ttemp));
		System.out.println("speed"+ UnicodeFormatter.byteToHex(btemp));
}

} //end class

/**
 * Package that will be transmitted over the RS-232 to Bluetooth device
 */
class JoyPackageJob extends Thread{

  private GUI myGUI;
  private Car myCar;
  private ControllLayer myStack;
  private byte[] dataByte;
  private int command;

 /**
  * Constructor
  *
  * @param GUI the grapichal interface
  * @param int command  1 = init
  *                     2 = inquiry
  *                     3 = createConnction
  */
 public JoyPackageJob(GUI theGUI, Car theCar,ControllLayer theStack,int theCommand){

		super();

		myGUI = theGUI;
		myCar=theCar;
		myStack=theStack;
		command=theCommand;

		//in case
		dataByte = new byte[1];
 }

 /**
  * Run method
  */
 public void run(){

		byte[] temp;
		boolean sucess;

		switch(command){

			//Init
			case 1 :  myCar.write("I -> Init BTcard at JoyPort...");
				  sucess=myStack.init();
				  myGUI.initCOM2Done(sucess);
				  myCar.setJoyInitialized(true);
```

```
                        break;

                //inquiry
                case 2 :  myCar.write("I -> Inquiry at JoyPort....");
                        sucess=myStack.inquiry();

                        //write error mesages
                        if(sucess){
                                myCar.write("I--> Inquiry sucess at JoyPort...");
                        } else{
                                myCar.write("E--> Inquiry failed at JoyPort...");
                        }
                        myGUI.inquiryDone();
                        break;

                //CreateConnection
                case 3 :  myCar.write("I -> Connecting to JoyStick...");
                        temp = myStack.createConnection(dataByte);
                        if(temp[0]==0x00 && temp[1]==0x00){
                                myCar.write("E --> New connection to joystick failed");
                                myGUI.connectionFailed(dataByte);
                        } else {
                                myGUI.connectionSuccess(dataByte);
                                myCar.myJoyPort.storeCID(temp);
                                myCar.myJoyPort.init=false;          //in case of error transmitting startvalue

                                byte[] joyData=new byte[11];
                                joyData[0]=(byte)0xff;      //analogout1=5V
                                joyData[1]=(byte)0x03;
                                joyData[2]=(byte)0xff;      //analogout2=5V
                                joyData[3]=(byte)0x03;
                                joyData[4]=(byte)0xfe;      //sets PWM period to 4.2 ms
                                joyData[5]=0x00;            //sets the PWM dutycycles to zero
                                joyData[6]=0x00;
                                joyData[7]=0x00;
                                joyData[8]=0x00;
                                joyData[9]=0x14;            //sets sampling sampling time=20ms
                                joyData[10]=0x00;                   //sets SPI off
                                myCar.myJoyPort.sendData(joyData);
                                myCar.setJoyConn(true);
                                if(myCar.getControlMode()== myGUI.stopMode){
                                        myGUI.setRightButtonMode();
                                }else{
                                        myGUI.setStopButtonMode();
                                }
                        }
                        break;

                //Error
                default:  myCar.write("E -> Trying to make illegal operation to controllLayer");
                        break;
        }

}

/*
* The data of the package are stored
*
* @param byte[] the data
*/
```

```
  public void storeData(byte[] data){

        dataByte = new byte[data.length];

        for(int i=0;i<data.length;i++){

              dataByte[i]=data[i];
        }

  }

}
```

----------Car----------

```
import java.util.Vector;
import java.awt.*;
import javax.swing.*;

public class Car{

	private ControllLayer com1stack,com2stack;
	private DebugWindow debug;
	private final int mouseMode=0, joyMode=1, stopMode=2;
	private int controlMode=stopMode, maxDistance, maxSpeed, mouseX, mouseY;
	private byte speed, steering;
	private boolean carInitialized,joyInitialized;
	private boolean carConn,joyConn,distanceControl;
	public final byte[] carAddress = new byte[6];
	public final byte[] joyAddress = new byte[6];
	private ControlThread t;


	public GUI myGUI;
	public JoyPort myJoyPort;
	public CarPort myCarPort;

	public static void main(String args[]){
		Car host = new Car();
	}

	/**
	* Constructor
	*/
	public Car(){
		debug=new DebugWindow();
		debug.setSize(new Dimension(850,500));
		debug.setLocation(0,500);
		myGUI = new GUI(this);
		myGUI.setSize(new Dimension(850,500));
		myGUI.setVisible(true);

		this.setAddresses();
		carConn=false;
		joyConn=false;
		myJoyPort = new JoyPort(this, joyAddress);
		myCarPort = new CarPort(this, carAddress);

		t = new ControlThread(this);
		t.start();
	}

	/***************************** Stack to Car *********************************/
	/*
	* Bluetooth unit discovered
	*
	* @param byte[] The address of the bluetooth unit
	*/
	public void newHost(byte[] address){
		myGUI.newHost(address);

	}
```

```
/************************** Application to stacks ****************************/
/*
* Send data to car
*
* @param byte[] The channel
* @param byte[] The data
*/
public void sendCarData(byte[] data){
	myCarPort.sendData(data);
}

/*
* Softwaer reset on carport
*/
public void resetCar(){

	if(myCarPort != null){
		myCarPort.reset();
	}
}

/*
* Software reset on joyport
*/
public void resetJoy(){

	if(myJoyPort != null){
		myJoyPort.reset();
	}

}

/*
* Send data to joystick
*
* @param byte[] The channel
* @param byte[] The data
*/
public void sendJoyData(byte[] data){
	myJoyPort.sendData(data);
}

/*
* init bluetooth card at com1
*
*/
public void com1init(){
	myCarPort.initComPort(1);
	myCarPort.initBT();
}

/*
* init bluetooth card at com2
*
*/
public void com2init(){
	myJoyPort.initComPort(2);
	myJoyPort.initBT();
}
```

```
/*
* Inquiry for bluetooth hosts with com1, the answer can be used by
* both com1 and com2.
*/
public void inquiry(){

	if(carInitialized){
		myCarPort.inquiry();
	}

	if(joyInitialized){
		myJoyPort.inquiry();
	}
}

/*
* Create connection with car
* @param byte[] bluetooth address
*/
public void createCarConnection(){
	myCarPort.createConnection(carAddress);
}

/*
* Create connection with joystick
*
* @param byte[] bluetooth address
*/
public void createJoyConnection(){
	myJoyPort.createConnection(joyAddress);
}

/*
* Close car connection
*
* @param byte[] The BT-address
*/
public void closeCarConnection(){
	myCarPort.closeConnection(carAddress);
}

/*
* Close joystick connection
*
* @param byte[] The BT-address
*/
public void closeJoyConnection(){
	myJoyPort.closeConnection(joyAddress);
}

/***************************** Other Car Methods ***************************/
/*
* Shows the DebugWindow, where onlyErrors decide whether Errors or
* Errors&Info should be shown
*
*/
public void openDebugWindow(boolean onlyErrors){
	debug.setInfoMode(onlyErrors);
	debug.setVisible(true);
	debug.repaint();
```

```
}

/*
*Writes a string to the DebugWindow.
*/
public void write(String s){

        debug.write(s);
}

/*
*Methods that return variables.
*/
public synchronized byte getSpeed(){
        return speed;
}
public synchronized byte getSteering(){
        return steering;
}
public synchronized int getControlMode(){
        return controlMode;
}
public synchronized byte[] getCarAddress(){
        return carAddress;
}
public synchronized byte[] getJoyAddress(){
        return joyAddress;
}
public synchronized boolean isCarInitialized(){
        return carInitialized;
}
public synchronized boolean isJoyInitialized(){
        return joyInitialized;
}
public synchronized boolean isJoyConn(){
        return joyConn;
}
public synchronized boolean isCarConn(){
        return carConn;
}

/*
*Methods that set variables.
*/
public synchronized void setAddresses(){
        carAddress[0]=(byte)0x17;
        carAddress[1]=(byte)0x28;
        carAddress[2]=(byte)0x03;
        carAddress[3]=(byte)0xb7;
        carAddress[4]=(byte)0xd0;
        carAddress[5]=(byte)0x00;
        joyAddress[0]=(byte)0x5c;
        joyAddress[1]=(byte)0x16;
        joyAddress[2]=(byte)0x03;
        joyAddress[3]=(byte)0xb7;
        joyAddress[4]=(byte)0xd0;
        joyAddress[5]=(byte)0x00;
}
public synchronized void setMaxSpeed(int sp){
        //maxSpeed=sp;
```

```
                myJoyPort.setMaxSpeed(sp);
        }
        public synchronized void setDistanceControl(boolean b){
                distanceControl=b;
                myJoyPort.addOne();
        }
        public synchronized void setMaxDistance(int di){
                maxDistance=di;
        }
        public  synchronized void setSpeed(byte sp){
                speed=sp;
        }
        public  synchronized void setSteering(byte st){
                steering=st;
        }
        public synchronized void setCarConn(boolean conn){
                carConn=conn;
        }
        public synchronized void setJoyConn(boolean conn){
                joyConn=conn;
        }
        public synchronized  void setControlMode(int mode){
                if(mode==stopMode){
                        t.firstTime=true;
                }
                controlMode=mode;
        }
        public synchronized void setCarInitialized(boolean i){
                carInitialized=i;
        }
        public synchronized void setJoyInitialized(boolean i){
                joyInitialized=i;
        }
}
```

```
----------------------------------------------ControlThread----------------------------------------------

public class ControlThread extends Thread{
	private Car myCar;
	private int controlMode;
	private final int mouseMode=0, joyMode=1, stopMode=2;
	private byte[] carData=new byte[11];
	public boolean firstTime=true;


	/*
	*Constructor
	*/
	public ControlThread(Car blue){
		myCar=blue;
		carData[0]=(byte)0x00;	//analogOut1=0V
		carData[1]=(byte)0x00;
		carData[2]=(byte)0x00;	//analogout2=0V
		carData[3]=(byte)0x00;
		carData[4]=(byte)0xff;	//sets maximum pulse period=23ms
		carData[5]=(byte)0xa1;	//This will make the dutycycle 1.5ms in pwm 1
		carData[6]=(byte)0x00;
		carData[7]=(byte)0x4a;	//This will make the dutycycle 1.5ms in pwm 2
		carData[8]=(byte)0x00;
		carData[9]=(byte)0xff;	//sets maximum sampling time=255ms
		carData[10]=(byte)0x00;	//sets SPI off
	}

	/*
	*The infinite loop
	*/
	public void run(){
		while(true){
			if(myCar.isCarConn()){
				controlMode=myCar.getControlMode();
				if(controlMode==stopMode){
					if(firstTime){
						//This will make the dutycycle 1.5ms
						carData[5]=(byte)0xA1;
						//This will make the dutycycle 1.5ms
						carData[7]=(byte)0x49;
						myCar.sendCarData(carData);
						firstTime=false;
					}
				}else{//joyMode or mouseMode
					carData[5]=myCar.getSteering();	//sets the speed
					carData[7]=myCar.getSpeed();	//sets the steering
					myCar.sendCarData(carData);
				}
			}

			try{
				sleep(100);
			} catch (Exception e){
				myCar.write("E -> ControlThread woke up during sleep! " + e.toString());
			}
		}
	}
}
```

---------- DebugWindow----------

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

   public class DebugWindow extends JFrame {
          private boolean justErrors=true;
          private JList eList,eiList;
          private DefaultListModel eModel, eiModel;
          private   JScrollPane ePane, eiPane;
          private Container contentPane;

   public DebugWindow() {
           super("Debug Information for the Bluetooth Connections");
           contentPane = getContentPane();

          eModel=new DefaultListModel();
          eModel.addElement("START OF COMMUNICATION");
          eList = new JList(eModel);
          ePane = new JScrollPane(eList);
          ePane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
          ePane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
          contentPane.add(ePane);

          eiModel=new DefaultListModel();
          eiModel.addElement("START OF COMMUNICATION");
          eiList = new JList(eiModel);
          eiPane = new JScrollPane(eiList);
          eiPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
          eiPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

           addWindowListener(new Cross(this) {
          });
}

public synchronized void setInfoMode(boolean onlyErrors){

          if (justErrors)
                    contentPane.remove(ePane);
          else
                    contentPane.remove(eiPane);
          justErrors=onlyErrors;
          if (justErrors)
                    contentPane.add(ePane);
          else
                    contentPane.add(eiPane);
}

public synchronized void write(String t){
          try {
                    t.trim();
                    eiModel.insertElementAt(t,0);
                    //eiModel.fireIntervallAdded(t,0,
                    if (t.charAt(0) == 'E' || t.charAt(0) == 'e'){
                              eModel.insertElementAt(t,0);
                    }

                    // remove old messsages
```

```
                    if(eModel.size()>=200){
                        try {
                            eModel.removeElement(eModel.lastElement());
                        } catch (Exception e){
                            System.out.println("Exception while removing error");
                        }
                    }

                    if(eiModel.size()>=200){
                        try {
                            eiModel.removeElement(eiModel.lastElement());
                        } catch (Exception e){
                            System.out.println("Exception while removing error");
                        }
                    }

                    contentPane.repaint();

            } catch (ArrayIndexOutOfBoundsException e){
                        System.out.println("Out of Bounds");
            } catch (Exception e){
                        System.out.println("Generall");
            }
}

public void repaint(){
        contentPane.repaint();
}

} // end class

class Cross extends WindowAdapter{
        DebugWindow debug;
        public Cross(DebugWindow d){
                debug=d;
          }

          public void windowClosing(WindowEvent e) {
                debug.setVisible(false);
        }
}
```

----------GUI----------

```
import javax.swing.table.TableColumn;
import javax.swing.DefaultCellEditor;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.JTable;
import javax.swing.JSlider;
import javax.swing.table.AbstractTableModel;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.JOptionPane;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.awt.Color;

//IO
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.comm.*;
import java.util.Calendar;

public class GUI extends JFrame implements ActionListener  {
            protected JMenuBar menuBar;
           private JMenuItem menuItem;
            private JMenu menu,menu2;
           private MyTableModel1 defaultModel;
           private GUIHostList myList;

           public Car myCar;
           public Container contentPane;
           public ImagePanel picturePanel;
           public JPanel bottom, btPanel, carPanel, com1Panel, com2Panel,connPanel;
           //public  MousePanel mousePanel;
           public JButton mouseButton, joyButton, stopButton, com1Button, com2Button;
           public JButton searchButton;
           public JCheckBox distance;
           public JSlider distanceSlider,speedSlider;
           public Hashtable labelTable;
           public JLabel text1, text2, text3, com1Field, com2Field;
           public DefaultListModel btModel;
           public JList btList;
           public JScrollPane btScroll;
           public JTable controllTable;

           private boolean inAnApplet,found;
           public final int mouseMode=0, joyMode=1, stopMode=2;

   public GUI(Car blue) {
           super("Taking Bluetooth One Step Furter");

           myCar=blue;
           menuBar = new JMenuBar();
            setJMenuBar(menuBar);
             myList=new GUIHostList();
```

```
 //Build the first menu.
 menu = new JMenu("File");
 menu.getAccessibleContext().setAccessibleDescription(
        "The only menu in this program that has menu items");
  menuBar.add(menu);

 //Build the next menus
menu2 = new JMenu("View");
menuBar.add(menu2);

 menuItem = new JMenuItem("Exit");
 menuItem.addActionListener(this);
 menu.add(menuItem);

 menuItem = new JMenuItem("Errors");
 menuItem.addActionListener(this);
 menu2.add(menuItem);

 menuItem = new JMenuItem("Errors & Info");
 menuItem.addActionListener(this);
 menu2.add(menuItem);

 // Create btpanal
 btPanel = new JPanel();
btPanel.setLayout(new BoxLayout(btPanel, BoxLayout.Y_AXIS));
btPanel.setBorder(BorderFactory.createLoweredBevelBorder());

// Create carpanel
carPanel = new JPanel(new GridLayout(0,2,15,5));
 carPanel.setBorder(BorderFactory.createLoweredBevelBorder());

 //          mousePanel=new MousePanel();  // is added at mouseControl
//           mousePanel.setSize(new Dimension(250, 100));
//           MyMouseListener m = new MyMouseListener(mousePanel,myCar);
//           mousePanel.addMouseMotionListener(m);   //movement

//Create picture panal with a car image
Image myImage = Toolkit.getDefaultToolkit().getImage("Beetfinal.gif");
picturePanel = new ImagePanel(myImage);
picturePanel.setSize(new Dimension(300, 200));

text1 = new JLabel("BLUETOOTH COMMUNICATION");
 text1.setFont(new Font("hej",Font.BOLD,17));
text1.setAlignmentX(CENTER_ALIGNMENT);
btPanel.add(text1);
com1Panel = new JPanel();
com1Panel.add(new JLabel("COM1:"));
com1Panel.add(com1Field = new JLabel(" Press init -->   "));
com1Field.setBorder(BorderFactory.createLoweredBevelBorder());
com1Panel.add(com1Button = new JButton("Init BT at COM1"));
com1Button.addActionListener(this);
btPanel.add(com1Panel);
com2Panel = new JPanel();
com2Panel.add(new JLabel("COM2:"));
com2Panel.add(com2Field = new JLabel(" Press init -->   "));
com2Field.setBorder(BorderFactory.createLoweredBevelBorder());
com2Panel.add(com2Button = new JButton("Init BT at COM2"));
com2Button.addActionListener(this);
btPanel.add(com2Panel);
btPanel.add(searchButton = new JButton("Search for BT-devices"));
```

```
searchButton.setAlignmentX(CENTER_ALIGNMENT);
searchButton.setEnabled(false);
searchButton.addActionListener(this);
text2 = new JLabel("Available BT-devices:");
text2.setAlignmentX(JLabel.CENTER_ALIGNMENT);
btPanel.add(text2);

// Create JTable
defaultModel = new MyTableModel1(this);
controllTable = new JTable(defaultModel);
controllTable.setPreferredScrollableViewportSize(new Dimension(400,200));
btScroll = new JScrollPane(controllTable);

//Fiddle with the Mode column's cell editors/renderers.
setUpModeColumn(controllTable.getColumnModel().getColumn(2));

//SetSize of table
TableColumn column = null;
for (int i = 0; i < 3; i++) {
        column = controllTable.getColumnModel().getColumn(i);
        if (i == 1) {
                column.setPreferredWidth(200); //Status column is bigger
        } else {
                column.setPreferredWidth(100);
        }
}

btPanel.add(btScroll);

/*
 btModel=new DefaultListModel();
 btList = new JList(btModel);
btScroll = new JScrollPane(btList);
btScroll.setBorder(BorderFactory.createLoweredBevelBorder());
btPanel.add(btScroll);
*/

text3=new JLabel("CONTROL OF RADIOCAR",JLabel.CENTER);
// text3.setHorizontalTextPosition(JLabel.CENTER);
// carPanel.setAlignmentX(CENTER_ALIGNMENT);
text3.setFont(new Font("hej",Font.BOLD,17));
carPanel.add(text3);
carPanel.setAlignmentX(CENTER_ALIGNMENT);
carPanel.add(new JLabel(""));
carPanel.add(mouseButton = new JButton("Steer with mouse"));
mouseButton.addActionListener(this);
speedSlider = new JSlider(6,30,30);
labelTable = new Hashtable();
labelTable.put( new Integer( 6 ), new JLabel("Minimum speed"));
labelTable.put( new Integer( 24 ), new JLabel("Maximum speed"));
speedSlider.setLabelTable( labelTable );
speedSlider.setPaintLabels(true);
speedSlider.addChangeListener(new SpeedSliderListener());
carPanel.add(speedSlider);
carPanel.add(joyButton = new JButton("Steer with joystick"));
joyButton.addActionListener(this);
carPanel.add(distance = new JCheckBox("Use distance control",true));
distance.addActionListener(this);
distance.setEnabled(false);
carPanel.add(stopButton = new JButton("Stop car"));
```

```
        stopButton.addActionListener(this);
        distanceSlider = new JSlider(10,30,30);
        labelTable = new Hashtable();
        labelTable.put( new Integer( 6 ), new JLabel("Minimum distance") );
        labelTable.put( new Integer( 24 ), new JLabel("Maximum distance") );
        distanceSlider.setLabelTable( labelTable );
        distanceSlider.setPaintLabels(true);
        distanceSlider.addChangeListener(new DistanceSliderListener());
        distanceSlider.setEnabled(false);
        carPanel.add(distanceSlider);

        this.setRightButtonMode();

        addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
        if (inAnApplet) {
                dispose();
         } else {

                // reset bluetooth devices
                myCar.resetCar();
                myCar.resetJoy();

                // sleep to allow proper disconnect
                //sleep(400);

                System.exit(0);

          }
        }
    });

        //Use the content pane's default BorderLayout.
        bottom = new JPanel(new BorderLayout(2,2));
        bottom.add(btPanel,BorderLayout.WEST);
        bottom.add(carPanel,BorderLayout.SOUTH);
        bottom.add(picturePanel,BorderLayout.CENTER);
        contentPane = getContentPane();
        contentPane.add(bottom);
        contentPane.repaint();

}

        class SpeedSliderListener implements ChangeListener {
                 public void stateChanged(ChangeEvent e) {
                        JSlider source = (JSlider)e.getSource();
                        myCar.setMaxSpeed((int)source.getValue());
                }
        }

        class DistanceSliderListener implements ChangeListener {
                public void stateChanged(ChangeEvent e) {
                        JSlider source = (JSlider)e.getSource();
                        myCar.setMaxDistance((int)source.getValue());
                }
        }

   public void setUpModeColumn(TableColumn modeColumn) {
        //Set up the editor for the sport cells.
        JComboBox comboBox = new JComboBox();
```

```
            comboBox.addItem("Connect");
            comboBox.addItem("Disconnect");
             //comboBox.showPopup();

             modeColumn.setCellEditor(new DefaultCellEditor(comboBox));

            //Set up tool tips for the sport cells.
            DefaultTableCellRenderer renderer =
            new DefaultTableCellRenderer();
            renderer.setToolTipText("Press button and choose command");
            modeColumn.setCellRenderer(renderer);

            //Set up tool tip for the sport column header.
            TableCellRenderer headerRenderer = modeColumn.getHeaderRenderer();
            if (headerRenderer instanceof DefaultTableCellRenderer) {
                        ((DefaultTableCellRenderer)headerRenderer).setToolTipText(
                        "Click below to see a list of choices");
            }
    }

// knapparnas händelsehanterare och meny
public void actionPerformed(ActionEvent e) {
            if (e.getActionCommand().equals("Errors & Info")) {
                        myCar.openDebugWindow(false);
              }else if (e.getActionCommand().equals("Errors")) {
                        myCar.openDebugWindow(true);
             }else if (e.getActionCommand().equals("Exit")) {

                         // reset bluetooth devices
                        myCar.resetCar();
                        myCar.resetJoy();

                        // sleep to allow proper disconnect
                        //sleep(400);

                        System.exit(0);

             }else if (e.getActionCommand().equals("Init BT at COM1")) {

                         com1Field.setText(" Init failed        ");
                         myCar.com1init();

             }else if (e.getActionCommand().equals("Init BT at COM2")) {

                         com2Field.setText(" Init failed        ");
                         myCar.com2init();

             }else if (e.getActionCommand().equals("Search for BT-devices")) {

                        //btModel.removeAllElements();
                        searchButton.setEnabled(false);
                        myCar.inquiry();

             }else if (e.getActionCommand().equals("Steer with mouse")) {

                        myCar.setControlMode(mouseMode);
                        this.setStopButtonMode();

             }else if (e.getActionCommand().equals("Steer with joystick")) {
                        myCar.setControlMode(joyMode);
```

```
                this.setStopButtonMode();
            }else if (e.getActionCommand().equals("Stop car")) {
                myCar.setControlMode(stopMode);
                this.setRightButtonMode();
            }else if (e.getActionCommand().equals("Use distance control")) {
                if (distance.isSelected()){
                    myCar.setDistanceControl(true);
                }else{
                    myCar.setDistanceControl(false);
                }
            }
    }

    /*
     * initMouseMode
     */

     /*
    public void initMouseMode(){
        myMousePanel.addMouseListener(mouseAdapter);
        myCarPanel.setSize(new Dimension(250, 100));
        MyMouseListener m = new MyMouseListener(myCarPanel,myList,myLogic);
        myCarPanel.addMouseMotionListener(m);   //movement
    }
*/

        /*
        * Update GUI after init on COM1
        */
        public void initCOM1Done(boolean sucess){

            if(sucess){

                com1Field.setText(" Init succeded   ");
                myCar.setCarInitialized(true);
                searchButton.setEnabled(true);

             }else{

                com1Field.setText(" Init failed      ");

             }
            com1Panel.updateUI();
        }

        /*
        * Update GUI after init on COM1
        */
        public void initCOM2Done(boolean sucess){

            if(sucess){

                com2Field.setText(" Init succeded   ");
                myCar.setJoyInitialized(true);
                searchButton.setEnabled(true);

             }else{

                com2Field.setText(" Init failed      ");
```

```
            }
            com2Panel.updateUI();
}

/*
* Update GUI after Inquiry. Enable new inquiry
*/
public void inquiryDone(){
            searchButton.setEnabled(true);
}

/*
* The method that is invoked when the stack finds a device;
*/
public void newHost(byte[] address){

            TableData temp;
            found=false;

            for(int i=0;i<myList.getSize();i++){

                        temp = (TableData) myList.getHostAt(i);
                        if(BlackBox.equals(address,temp.getAddress())){
                                    found= true;
                                    break;
                        }
            }

            if(!found){
                        byte[] carAddress;
                        byte[] joyAddress;

                        carAddress=myCar.getCarAddress();
                        joyAddress=myCar.getJoyAddress();

                        if(BlackBox.equals(address,carAddress)){
                                    temp = new TableData("Radiocar");
                                    temp.setAddress(address);
                                    temp.setStatus("Not connected");
                                    myList.storeHost(temp);
                                    defaultModel.update(temp);
                        } else if(BlackBox.equals(address,joyAddress)){
                                    temp = new TableData("Joystick");
                                    temp.setAddress(address);
                                    temp.setStatus("Not connected");
                                    myList.storeHost(temp);
                                    defaultModel.update(temp);
                        } else {
                                    temp = new TableData("Unknown device");
                                    temp.setAddress(address);
                                    temp.setStatus("Not connected");
                                    myList.storeHost(temp);
                                    defaultModel.update(temp);
                        }
            }
}
```

```
/*
* The method that is invoked when the stack finds a device;
*/
public void connect(String device,TableData temp){

        if(device.equals("Radiocar")){

                if(myCar.isCarInitialized()){
                        myCar.createCarConnection();
                        temp.setStatus("Trying to connect");
                } else {
                        temp.setStatus("Radiocar can only connect to COM1");
                }

        } else if(device.equals("Joystick")){

                if(myCar.isJoyInitialized()){
                        myCar.createJoyConnection();
                        temp.setStatus("Trying to connect");
                } else {

                        temp.setStatus("Joystick can only connect to COM2");

                }

        } else {

                temp.setStatus("Can't connect to unknown device");

        }
 }

/*
*
*/
public void writeConnected(byte[] address){

        TableData temp =null;
        int i=0;

        for(;i<myList.getSize();i++){

                temp = (TableData) myList.getHostAt(i);
                if(BlackBox.equals(address,temp.getAddress())){
                break;
                }
        }

        if(i<myList.getSize()){

                if(temp.getDevice().equals("Radiocar")){
                        temp.setStatus("Connection reestablished");
                } else if(temp.getDevice().equals("Joystick")){
                        temp.setStatus("Connection reestablished");
                }

                defaultModel.updateAll();
        }
}
```

```
	/*
	* The method that is invoked when the stack finds a device;
	*/
	public void disconnect(String device,TableData temp){

		if(device.equals("Radiocar")){

			if(myCar.isCarConn()){
				myCar.closeCarConnection();
				temp.setStatus("Disconnected");
			} else {
				temp.setStatus("Car is not connected");
			}

		} else if(device.equals("Joystick")){

			if(myCar.isJoyConn()){
				myCar.closeJoyConnection();
				temp.setStatus("Disconnected");
			} else {
				temp.setStatus("Joystick is not connected");
			}

		} else {
			temp.setStatus("Can't disconnect unknown device");
		}
}

	/*
	 * Method updates GUI if connection is closed by other host
	*/
	public void connectionClosed(byte[] address){

		TableData temp =null;
		int i=0;

		for(;i<myList.getSize();i++){

			temp = (TableData) myList.getHostAt(i);
			if(BlackBox.equals(address,temp.getAddress())){
				break;
			}
		}

		if(i<myList.getSize()){
			temp.setStatus("Connection closed by other host");
			temp.setCommand("Disconnect");
			defaultModel.updateAll();
		}
	}

	/*
	 * Metod updates GUI if a attemp to connect fail
	 */
	 public void connectionFailed(byte[] address){

		TableData temp =null;
		int i=0;

		for(;i<myList.getSize();i++){
```

```
                temp = (TableData) myList.getHostAt(i);
                if(BlackBox.equals(address,temp.getAddress())){
                break;
                }
        }

        if(i<myList.getSize()){
                temp.setStatus("Connection failed");
                temp.setCommand("Disconnect");
                defaultModel.updateAll();
        }
    }

 /*
  * Method updates GUI if a attemp ro connect is a success
 */
public void connectionSuccess(byte[] address){

        TableData temp =null;
        int i=0;

        for(;i<myList.getSize();i++){

                temp = (TableData) myList.getHostAt(i);
                if(BlackBox.equals(address,temp.getAddress())){
                break;
                }
        }

        if(i<myList.getSize()){
                temp.setStatus("Connected");
                temp.setCommand("Disconnect");
                defaultModel.updateAll();
        }
}

/*
* Below are methods that enables different buttons.
*/
public void setStopButtonMode(){
        stopButton.setEnabled(true);
        joyButton.setEnabled(false);
        mouseButton.setEnabled(false);
}
public void setMouseButtonMode(){
        stopButton.setEnabled(false);
        joyButton.setEnabled(false);
        mouseButton.setEnabled(false);// set this to true when mouse is implemented!!!!!!!!!!!!!!!!!
}
public void setJoyButtonMode(){
        stopButton.setEnabled(false);
        joyButton.setEnabled(true);
        mouseButton.setEnabled(false);// set this to true when mouse is implemented!!!!!!!!!!!!!!!!!
}
public void setNoButtonMode(){
        stopButton.setEnabled(false);
        joyButton.setEnabled(false);
        mouseButton.setEnabled(false);
}
```

```
	public void setRightButtonMode(){
		if(myCar.isCarConn()){
			if(myCar.isJoyConn()){
				setJoyButtonMode();
			}else{
				setNoButtonMode();
			}
		}else{
			setNoButtonMode();
		}
	}

} // end class

/************************** Image Panel*****************************************/
class ImagePanel extends JPanel {
	Image image;

	public ImagePanel(Image image) {
		this.image = image;
	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g); //paint background

		//System.out.println("Painting image");

		 if(image == null)
			System.out.println("E -> Cant Display image");

		 //Draw image at its natural size first.
		g.drawImage(image, 0, 0, this); //85x62 image

		 //Now draw the image scaled.
		 //g.drawImage(image, 90, 0, 300, 62, this);

	 }
}

/************************** Image Panel*****************************************/
class MousePanel extends JPanel {

	public MousePanel() {
	}

	 public void paintComponent(Graphics g) {
		 super.paintComponent(g); //paint background
	}
}

/*********************************Table Modell *********************************/

/**
 * Model for JTable
 */
class MyTableModel1 extends AbstractTableModel {
   protected static int NUM_COLUMNS = 3;
   protected static int START_NUM_ROWS = 0;
   protected int nextEmptyRow = 0;
   protected int numRows = 0;
```

```
    private GUI myGUI;

    final String[] columnNames = {"Device","Status","Command"};

    protected Vector data = null;

    public MyTableModel1(GUI theGUI) {
              data = new Vector();
              myGUI=theGUI;
    }

    public int getNumberOfRow(){
            return numRows;
    }

    public int getColumnCount() {
            return columnNames.length;
    }

    public int getRowCount() {
            if (numRows < START_NUM_ROWS) {
                      return START_NUM_ROWS;
            } else {
                      return numRows;
            }
    }

    public String getColumnName(int col) {
            return columnNames[col];
    }

    public Object getValueAt(int row, int col) {
            try {
               TableData c = (TableData) data.elementAt(row);
               switch (col) {
                      case 0: return c.getDevice();
                      case 1: return c.getStatus();
                      case 2: return c.getCommand();
               }

            } catch (Exception e) {
            }
            return "";
    }

  /**
     * Don't need to implement this method unless your table's
     * editable.
     */
    public boolean isCellEditable(int row, int col) {

                      if(col >= 2){
                              return true;
                      } else {
                              return false;
                      }

    }
```

```
	public void setValueAt(Object value, int row, int col) {

		TableData temp;
		int location;

		location=0;
		temp = (TableData) data.elementAt(row);

		switch (col) {

		case 0:	temp.setDevice((String) value);
			break;

		case 1:	temp.setStatus((String) value);
			break;

		case 2:	String temp2;
			temp2 = (String) value;

			// change panel
			if(temp2.equals("Disconnect")){
				myGUI.disconnect(temp.getDevice(),temp);
				temp.setCommand("Disconnect");
				updateAll();
			} else if(temp2.equals("Connect")){
				myGUI.connect(temp.getDevice(),temp);
				temp.setCommand("Connect");
				updateAll();
			}
			break;
		}

		 fireTableCellUpdated(row, col);
}

/*
 * JTable uses this method to determine the default renderer/
 * editor for each cell.  If we didn't implement this method,
 * then the last column would contain text ("true"/"false"),
 * rather than a check box.
 */
public Class getColumnClass(int c) {
	return getValueAt(0, c).getClass();
}

// stoppa in object av klassen number Data
public void update(TableData n){

	numRows++;
	data.addElement(n);
	nextEmptyRow++;
	fireTableDataChanged();

}

/*
* repaint on all table
*/
public void updateAll(){
```

```
            fireTableDataChanged();
    }


     public void delete(String device){

            TableData t;

            for (int i=0;i<data.size();i++){
                    t = (TableData) data.elementAt(i);

                    if(device.equals(t.getDevice())){
                            data.removeElementAt(i);
                            nextEmptyRow--;
                            numRows--;
                            fireTableDataChanged();
                    }
            }
     }

}

/************************** TableData*****************************************/

/**
 *  Class of the data that id stored in JTable
 */
class TableData{

  private String myDevice;                  // description
  private String myCommand;                 // mode
  private String status;
  private byte[] address;                   // BT adress

  public TableData(String theDevice){

            myDevice = theDevice;
            myCommand = "Disconnect";
            status="Not connected";

            address = new byte[6];
  }

  /************************** GET OPERATION ****************************************/
  public String getDevice(){
            return myDevice;
  }

   public String getCommand(){
            return myCommand;
  }

    public String getStatus(){
            return status;
  }

  public byte[] getAddress(){
            return address;
  }
```

```
  /*************************** SET OPERATION ****************************************/

  public void setDevice(String s){
          myDevice =s;
  }

  public void setCommand(String s){
          myCommand=s;
  }

   public void setStatus(String s){
          status=s;
  }

  public void setAddress(byte[] t){

          address[0]=t[0];
          address[1]=t[1];
          address[2]=t[2];
          address[3]=t[3];
          address[4]=t[4];
          address[5]=t[5];
  }

}

/**********TableData, Storage Place for Bluetooth connections ************************/

/**
 * Class containing a accesible list of all
 * bluettoth devices.
 */
class GUIHostList{

          private Vector myVector;

          /**
            * Constructor
           */
          public GUIHostList(){
                  myVector = new Vector();
          }

          /**
           *  Return á stored GUIHost object
           *
           * @param byte[] The BT adress
           * @return The found TableData object otherwise null
           */
          public synchronized TableData getHost(byte[] adress){

                  TableData temp;

                  for (int i=0;i<myVector.size();i++){
                          temp = (TableData) myVector.elementAt(i);
                          if(BlackBox.equals(adress,temp.getAddress())){
                                  return temp;
                          }
                  }
```

```
                return null;
        }

        /**
         * Return TableData at index
         *
         * @param int index
         * @param A stored TableData
         */
         public synchronized TableData getHostAt(int index){

                if(index < 0 || index >= myVector.size()){
                        return null;
                }

                return (TableData) myVector.elementAt(index);
         }

        /**
         * Store TableData object in class
         *
         * @param TableData The object
         */
        public synchronized void storeHost(TableData s){
                myVector.addElement(s);
        }

        /**
         * Return number of stored TableDatas
         *
         * @param Stored TableDatas
         */
        public synchronized int getSize(){
                return myVector.size();
        }

        /**
         * Remove TableData from class
         *
         * @param The TableData that will be removed
         */
        public synchronized void removeHost(TableData s){
                myVector.remove(s);
        }
}
```

Appendix F: Technical Information about Bluetooth

F.1 General Information

Bluetooth is an industry standard for wireless connection between electrical devices. The idea is that this chip should be implemented in all possible devices such as for example computers and cellular phones. It will then be very easy to exchange information and a whole new world of possibilities will arise, such as cheap wireless headsets and ad hoc computer networks. But are there not already wireless systems today and why would Bluetooth be better than any other system?

Well, this technology is made to be very cheap and compatible between all kinds of devices. Bluetooth is a technology specification that has been developed by the Bluetooth Special Interest Group, which consists of the leaders in telecommunication, computing, and network industries. These are 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia and Toshiba. The standard is also supported by 2000 adopter companies. It is therefore very likely that this specification will be accepted. While so many companies have unified, it has become the fastest growing industry standard ever. A big advantage with the Bluetooth chip is that when it has been produced in a larger volume it is predicted to cost 5$. It will therefore be possible to replace cables with this technology just because it is cheaper than cables. Another great advantage is that it will be able to transmitt both a voice and data, which makes the chip interesting for all kinds of devices. Personal ad hoc networks can also simply be set up and the users will get instant access to each other's data. In order for this standard to succeed though, the technology must be good enough.

Ericsson in Lund has implemented one of the few applications that exists. It is a wireless headset for cellular phones. Another powerful application that could be realized is the three-in-one phone. That means that a person can use his or her cellular phone at home as a cordless phone, outdoors as usual, and at work both as a cordless phone and an internal phone. When the person is inside he or she needs a Bluetooth connection either to some device that transmits the call to the telephone net or to a computer. In the latter case IP-telephony could be used and the call would be transmitted over Ethernet, which then will make the phone call very cheap. This could of course be combined with the wireless headset and then both devices can be utilized. The Bluetooth chip in the cellular phone also makes it possible to connect to other cellular phones and exchange data, for example can business cards or information about a meeting be transmitted. Soon there will be a huge amount of cellular phones that can surf on the Internet and when possible the phone can connect to a computer via Bluetooth and surf for free. One incredible application is that personal ad hoc computer nets can be set up instantly and files can be sent to each other easily. Imagine that all the students in a course have laptops and when they enter the classroom they can all connect to the teacher's computer. The students can get any kind of information about the course on their computers and they can also respond and book laboratory time immediately. Suppose a customer's cellular phone connects to the computer net at an airport. Directly when the customer comes inside the airport the airport's computer net can call him or her via the net and tell that the flight is delayed or show how to get to the plane. The possibilities are enormous when Bluetooth chips will be installed everywhere. Instead of keys the cellular phone opens the doors people can access and the computer registers directly when the workers come and go from work. Handheld scanners, mobile hard disks, and other handheld devices will be possible to make to a low cost.

As seen there are an infinite number of applications of this chip. The technology might have some shortcomings, but it definitely has the best possible support from the industry. The Special Interest Group believes that this chip will be built into more than a billion devices in a couple of years. Bluetooth 2 is also on its way, with features such as ten times higher bit rate and range, then the scope of applications will be even bigger. Therefore we should prepare for a new world with Bluetooth in all possible digital devices. The great Viking Harald Bluetooth made whole Denmark christian. Now this tiny chip is here to revolutionize our IT-world!

F.2 Technical Information

The information below is according to the Bluetooth specification 1.0B [2]. All the functionality in the specification is though not supported by today's Bluetooth module from Ericsson. The technology consists of a tiny chip with a radio transceiver, which is built into digital devices. The chip can be used for both voice and data communication. For the radio communication, Bluetooth uses the free frequency band between 2.400-2.4835 GHz. These frequencies can be disturbed by for example microwave ovens. For this reason the frequency range is divided into 79 different 1 MHz bands, which the chip jumps between 1600 times per second. If the transmission fails in one frequency, the data can be retransmitted in another frequency. In each frequency GFSK (Gaussian Frequency Shift Keying) is used. This seems as a strange choice to us, as QAM has a better power to bandwidth relation and could have been used instead.

The different types of packages that can be sent are shown in Figure 1. The maximum bit rate is 723 kbps in one direction. The packages have different advantages. The slower packages are for instance smaller and safer to send.

type	symmetric	asymmetric	
DM1	108.8	108.8	108.8
DH1	172.8	172.8	172.8
DM3	258.1	387.2	54.4
DH3	390.4	585.6	86.4
DM5	286.7	477.8	36.3
DH5	433.9	723.2	57.6

[kbps]

Figure 1: The different packages that can be used and the speed they have. Symmetric and asymmetric dataflow can be chosen. When the highest data rate is chosen the Bluetooth chip uses five consecutive time slots and this is illustrated in Figure 2.

In order to get the asymmetric or the high bit rate the packages are sent in multi slots as can be seen in Figure 2. The large package at the bottom of the figure is sent during five time slots.

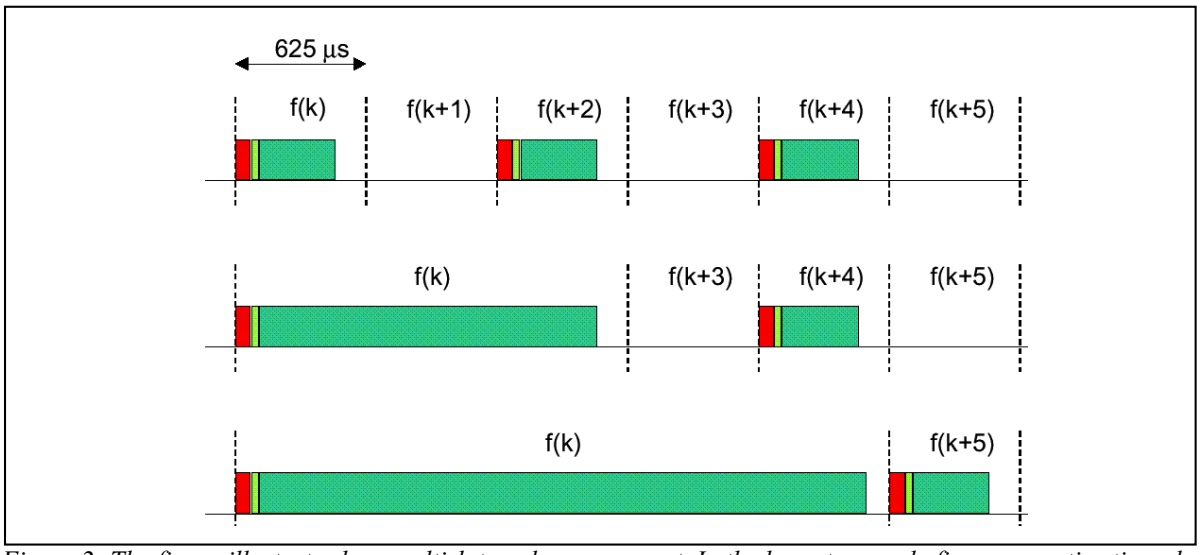


Figure 2: The figure illustrates how multislot packages are sent. In the lowest example five consecutice time slots are used in order to get the highest data rate.

The high data speed is enough for many devices but not for example a video recorder, which needs at least five times greater bit rate.

The radio output power is 1 mW maximum and therefore the maximum operating range becomes 10 m. Note that the devices do not have to be in the line of sight, because radio waves are used, so the devices can for example be in different rooms. The receiver has a sensitivity level of at least –70 dBm, which makes the bit-error-rate become 0.1% at this power level.

There can be up to eight Bluetooth units in a piconet. In such a net there are one master and seven slaves, where the master controls all the communication. Every Bluetooth chip has a unique address, which makes it possible to send a message to a certain unit. The master and slave communication seems to have been developed to have one “intelligent” node, for example a computer with a human, and several less intelligent. Therefore the slaves can not communicate with each other directly, unless the master addresses the communication. Slave-to-slave communication is though possible but it demands higher level protocols than described in the Bluetooth Specification. The master and the slaves in one piconet can participate as slaves in other piconets. Several piconets that are connected to each other form a scatternet. In Figure 3 an example shows how two piconets form a scatternet.

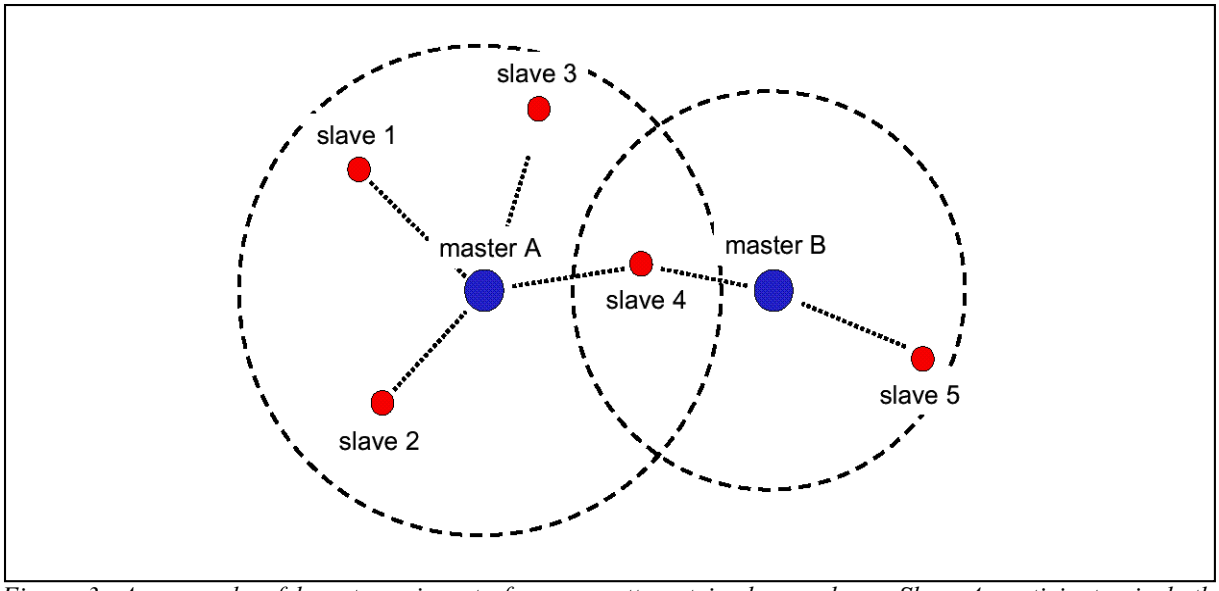


Figure 3: An example of how two piconets form a scatternet is shown above. Slave 4 participates in both piconets and can forward information from one piconet to the other.

The communication can either be synchronous or asynchronous. The standard can support either an asynchronous data channel, up to three synchronous voice channels, or one simultaneously synchronous voice and asynchronous data channel. The packets can be protected by error correcting code and they may also be encrypted. The Bluetooth devices can be in a couple of different modes, so that a device for example can participate in different piconets or to save power. As seen above, the specification of the standard is quite flexible in order to make the chip fit in many applications.

F.3 The Bluetooth specification 1.0 B stack

The Bluetooth stack can be seen in Figure 4. It is organised into different layers according to the OSI model [9]. The functionality of each layer is defined in a specific protocol in the Bluetooth specification 1.0 B [2]. The layers provide well-defined services to the above layers. The profiles specify different user applications that are predefined in the specification.

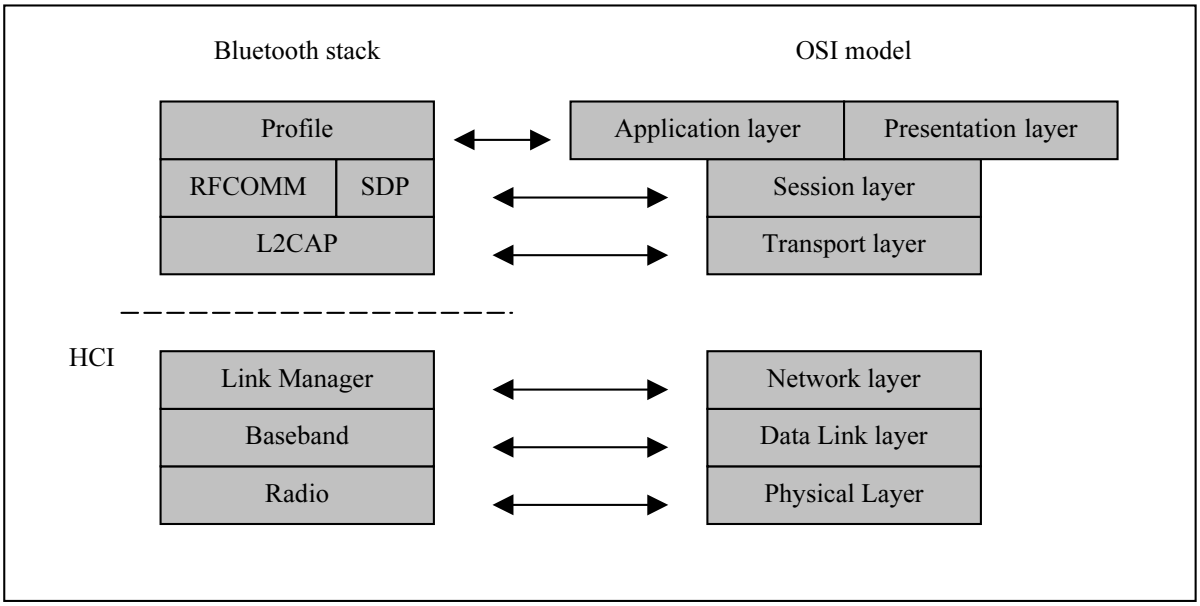


Figure 4: The Figure shows the Bluetooth stack and its relation to the OSI model [9]. The layers have all well seperated functionality and provide the above layer with Bluetooth independent services.

- Radio
  The radio operates in the unlicensed ISM band at 2.4 GHz. The data transmitted is frequency jumping at the rate of 1600 hop per second and has a symbol rate of 1 Mbit/s. A Gaussion-shaped, binary FSK modulation is applied with a BT product of 0.5.

- Baseband
  The baseband is the link between the radio and link manager. It performs services as error coding and other low-level link routines.

- Link Manager
  The link manager is used for link set-up, security and control.

- Host Controller Interface, HCI
  The HCI provides a uniform interface method for accessing the Bluetooth hardware capabilities.

- Logical Link Control and Adaptation Protocol, L2CAP
  L2CAP provides connection data services to upper layer protocols with multiplexing capability, segmentation and reassemble operation. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

- RFCOMM
  The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. RFCOMM is a simple transport protocol, with additional provisions for emulation of the 9 circuits of RS-232 (EIATIA-232-E) serial port.

- Service Discovery Protocol, SDP
  The SDP provides a means for applications to discover which services are available and to determine the characteristics of those available services. The services is specified in the profiles e.g. fax profile and file transfer profile

- Profile
  The profiles are predefined applications and are described in the text below. Different Bluetooth devices can support different profiles.

The different profiles of the Bluetooth specification 1.0 B and their relations can be seen in Figure 5. They are divided into three major groups. Generic Access Profile (GAP), Serial Port Profile and Generic Object Exchange Profile. All of the profiles in the specification must support the Generic Access Profile (GAP). The profiles in the serial port profile must as well support the serial port profile, and the profiles in the Generic Object Exchange Profile must support all three.

Products based on functionality not supported by the predefined profiles, can be developed and qualified as long as the Generic Access Profile is supported. The whole idea with profiles is that products developed by different developers still can communicate with each other.

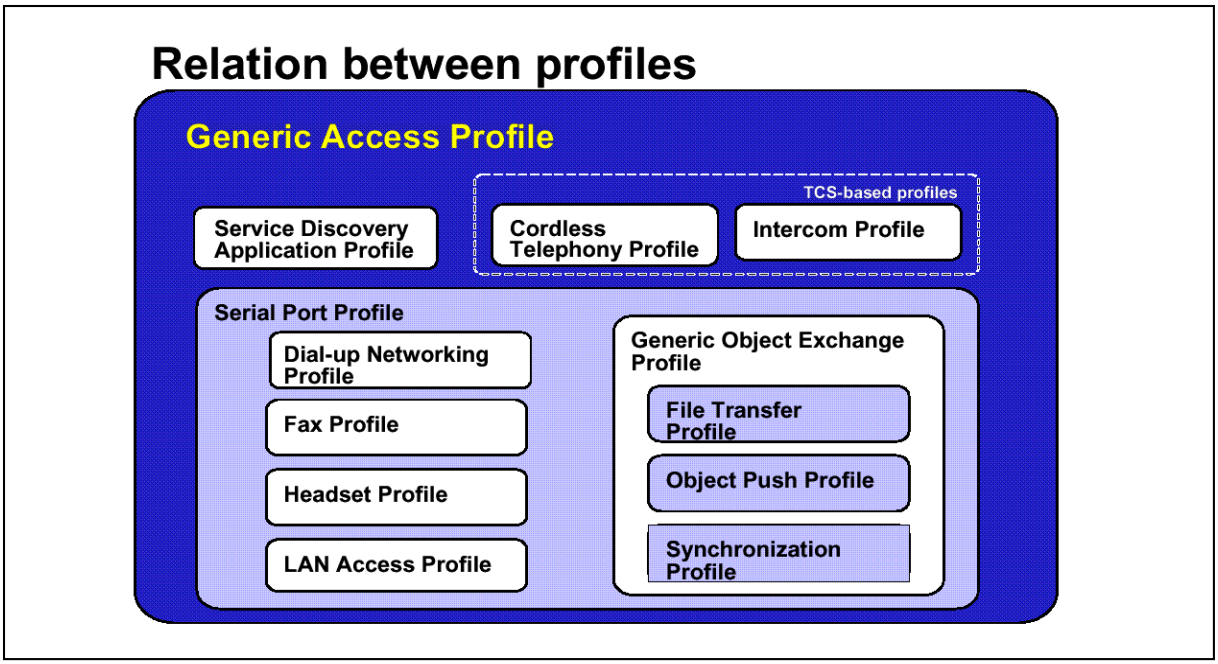


Figure 5: The Figure illustrates the different profiles and there relationship. All profiles must implement the Generic Access Profile. The profiles from the Serial Port Profile group must also implement the Serial Port Profile. The profiles in the Generic Object Exchange Profile group must implement all three. By forcing developers to support profiles the Bluetooth products are made compatible with each other.

- Generic Access Profile
  This profile defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices.

- Serial Port Profile
  This profile defines the requirement for Bluetooth devices necessary for setting up emulated serial cable connections using RFCOMM between two peer devices.

- Generic Object Exchange Profile
  This profile defines the requirements for Bluetooth devices necessary for the support of an general object exchange usage model. The requirements are expressed by defining the features and procedures that are required for interoperability between Bluetooth devices in the object exchange usage model.

F.4 Qualification

All Bluetooth products need to be tested and qualified according to the Bluetooth specification. The qualification is needed in order for the Bluetooth standard to succeed and to keep all Bluetooth products compatible with each other. If a company produces or trades a product that does not comply with the specification or that has not completed the qualification, the Bluetooth Special Interest Group (SIG) and all adopter companies will sanction that company. It is therefore important to know how the qualification program works and what it demands.

The Bluetooth Qualification Program is reviewed, managed, and improved by the Bluetooth Qualification Review Group (BQRG). The program establishes the rules and procedures how the manufacturer shows that their product complies with the specification, and how

manufacturers and distributors may use the Bluetooth license. The qualification is done and maintained by a Bluetooth Qualification Test Facility (BQTF), a Bluetooth Qualification Body (BQB), and a Bluetooth Qualification Administrator (BQA). These three are authorized by BQRG. Both the BQTF and BQB can be in-house functions in a company. When a company wants its product qualified it first hands it to the BQTF that tests the product. When the tests are done the test results plus the declarations and documentation are sent to a BQB. If the BQB sees that everything complies with the specification, he or she sends information about the new product to the BQA. The BQA puts the product into the Qualified Products List that can be viewed by everyone. The BQA also maintains documents for the Qualification Program so the manufacturers can get informed about the Qualification Process.

What is then demanded by the Qualification Program? Well, the product requirements are divided into radio link, protocol, profile, and information requirements. The radio link shall meet certain minimum requirements so the Bluetooth technology can assure a certain quality. The lower layers' protocols of the Bluetooth stack are tested with the Bluetooth Test Control Interface. This checks that the LC, LM, L2CAP and the HCI are complying with the specification. The BQRB will though authorize BQTFs that are allowed to qualify products against the protocol requirements. On top of these layers are there different profiles. Examples of these profiles are the synchronization profile and the headset profile. The following points describe the profile requirements:

- All products must follow the General Access profile.
- All implemented Bluetooth services must be described in the "Implementation Conformance Statement".
- All profiles that the manufacturer has declared in the "Implementation Conformance Statement" must be implemented according to each profile specification.
- If a service, for which there exists a Bluetooth profile, shall be implemented, it must be done according to that profile. It is permitted to make improvements or add features to a profile, as long as interoperability is maintained with other products that have implemented the standard. When extra features are added, these must be negotiated between the Bluetooth devices when they connect to each other.

The demand that all products must implement the General Access profile assures that all Bluetooth devices can communicate with each other. In case the devices do not have any common services they will at least get that information from each other. Finally, there are certain requirements on the information about the product. The information should be clear and consistent and should contain all the capabilities that are implemented in the device.

F.5 Competitors and Future Development of Bluetooth

Bluetooth does of course have a lot of competing technologies and the future depends on which technology that is the best and first reaches the market. The technology of Bluetooth does not excel in any special area; the thing is that there is no other technology that covers as many areas as Bluetooth.

However, the technologies that compete with Bluetooth within the radio-lan area are Hyperlan, IEEE 802.11, and Ultra Wideband (UWB). The American IEEE [5] will during 2001 choose one of the four radio techniques for the new standard Wireless Personal Area Network (WPAN). This standard is supposed to support a data speed at 20 to 50 Mbit/s within a range of about 10 meters. It is intended to be used for home nets and to simultaneously

handle three video channels and five voice channels. It should also be used for telephony and Internet. Today's version of Bluetooth does not have higher speed than 1 Mbit/s, but the next version of Bluetooth will have a speed around 10 Mbit/s. The 802.11 [6] has the capacity 11Mbit/s and is the strongest competitor to Bluetooth for the WPAN standard. UWB handles the capacity demands in WPAN best, but it has problems to cope with the frequency regulations in USA, Europe, and Asia.

Another area the Bluetooth standard aims for is the office environment with mice, printers, keyboards, and all the devices around the computer. The competitors in this case are infrared and other cheap radio based solutions. The Bluetooth chip will only cost $5 if it will be produced in a larger scale. The standard is therefore likely to succeed in this area. Bluetooth can be used in many areas where there is a need for wireless voice and data communication. The most conspicuous features of the Bluetooth standard compared to other technologies will be that it is cheap and has built in support to be compatible.

The Bluetooth specification has had some errors so far [7], therefore no products have been released on the market. Version 1.1 of the specification that comes in November 2000 is said to work correctly. The Bluetooth Special Interest Group (SIG) has strongly recommended all adopters not to release any products based on former versions of the specification. One threat to the Bluetooth standard is that there will continue to be errors in the specification. Another threat might be that it will be time and cost consuming to qualify products [8].

Bluetooth 2 is on its way and will be finished during 2001. In this version the data speed will be around 10 Mbit/s and the range up 100 meters. In this specification there will also be more profiles, which will make the standard even more general. The SIG, which consists of the nine leading companies in the telecommunications, computing, and network industries, is driving the development of the technology and bringing it to the market. That is an important fact that speaks in favor of Bluetooth.

Appendix G: Press release

http://207.94.167.208/keynoteBTeveryday.cfm#2

Tuesday, December 5
9:15 AM - 10:00 AM
Bluetooth Everyday

At Bluetooth Everyday you will watch someone perform everyday tasks in extraordinary ways using Bluetooth wireless technology. With no technical experience, you will see the demonstrator interact with Bluetooth in order to make life easier. Here's a glimpse of the demos you'll see:

The Bluetooth Controlled Beetle
This demonstration shows a radio control car being driven by joystick. Input from the joystick is transmitted using Bluetooth wireless technology to a computer that generates control signals that are sent to the car also using Bluetooth. The platform used - developed by Sigma with the Bluetooth Application Tool Kit - is generic and can also be used in a wide range of industrial processes.
By Sigma Comtec

As you walk into a shopping mall with your PDA you will receive special offers that are customized to where you are and who you are. You can easily accept the offer and perform the transaction.
By Axis Communications

XyLoc conveniently protects and personalizes a laptop utilizing Bluetooth wireless technology: a user will approach the PC and XyLoc will automatically unlock the PC, log the user into his accounts, decrypt his files, and open his applications. When the user steps away from the PC it will automatically be secured, preventing unauthorized access.
By Ensure Technologies

Its Monday morning in the Bradford residence and Tim is headed out to Los Angeles this morning for an important customer meeting. Tim comes down to breakfast in his kitchen. While eating his cereal, he uses his "InfoPad", a 8.5x11 paper-sized wireless information appliance, to log on to the Internet (using Bluetooth to dial out through the PC in his basement home office), in order to check on his flight and the weather at LAX, so he knows if he has to bring a jacket or a rain coat. While online he quickly notes that QCOM stock price is down and puts in an order to buy 100 shares.
By Impulsesoft

With the Motorola Timeport phone, which utilizes Bluetooth wireless technology, purchase gas and coffee at the gas station. While the gas is pumping, the Bluetooth Internet Server at the pump sends ads to the Timeport. The user can browse gifts from the gas vendor's Intranet and can also get directions on his Timeport.
By Motorola

Phone as remote control
The demo shows how you, via your Bluetooth mobile phone from Ericsson, can control various devices in your home environment by using WAP (Wireless Application Protocol) over Bluetooth wireless technology. In this particular demo the Bluetooth phone is used to turn a lamp on and off.

Bluetooth Headset
Some research analysts claim the Bluetooth Headset will become the most popular product that utilizes Bluetooth wireless technology. The Bluetooth Headset is connected to a compatible mobile phone. The user can either receive or make phone calls. Voice dialing is also possible. Not only will these products be demoed; they are made available by Ericsson.

Anoto pen
This pen brings the human pen-paper behavior to Bluetooth wireless technology. You use the Anoto pen to write normally on any paper that has a unique (almost invisible) pattern printed on it. The text is then wirelessly transferred via your Bluetooth phone. Imagine the e-commerce possibilities!
By Ericsson

WAP over Bluetooth™ Demo
AU-System shows how to control PC-applications such as CD-player, MP3-player etc. from a Palm Pilot. The demo will show how Bluetooth combined with WAP can enhance comfort in an office or home network. The PDA with Bluetooth wireless technology can control different devices without changing its interface.
By AU Systems

DENSO BLUETOOTH CONNECTIONS "wireless car kit function"
This demo will illustrate a Bluetooth wireless synchronous link between a mobile phone and an automotive hands free carkit. The user will be able to place a call with the phone, place it anywhere within 10m of the car kit and carry on a conversation through the carkit.
By Denso

See how SPANworks 2000 can enable you to effortlessly transfer files to one, a few, or everyone in your proximity with a simple drag 'n drop. Give a slide presentation among several machines, with or without the use of a digital projector. Or, "whisper" among friends using the chat messaging Application.
By Toshiba

Appendix H: Performance and characteristics of the BCC

The Bluetooth Control Card (BCC) can set and measure analog signals in the region of 0-5 V. It has also a digital Serial Peripheral Interface (SPI), which can receive and transmit 6 bytes of data in each slave to master package. The sample period is user defined in the interval 6-255 milliseconds and the outputs are updated when a valid data package is received.

The BCC has 8 different input and outputs:

- Analog in 1
- Analog in 2
- Analog out 1
- Analog out 2
- PWM out 1
- PWM out 2
- SPI
- Digital interrupt

The general performance of the Bluetooth technology is discussed in Appendix F Section 2.

Figure 1 shows the delay from slave to master for a sampled input signal, before it is presented to the API in the master. It has been measured to ~ 15 milliseconds. The same delay is valid for output signals from master to slave.

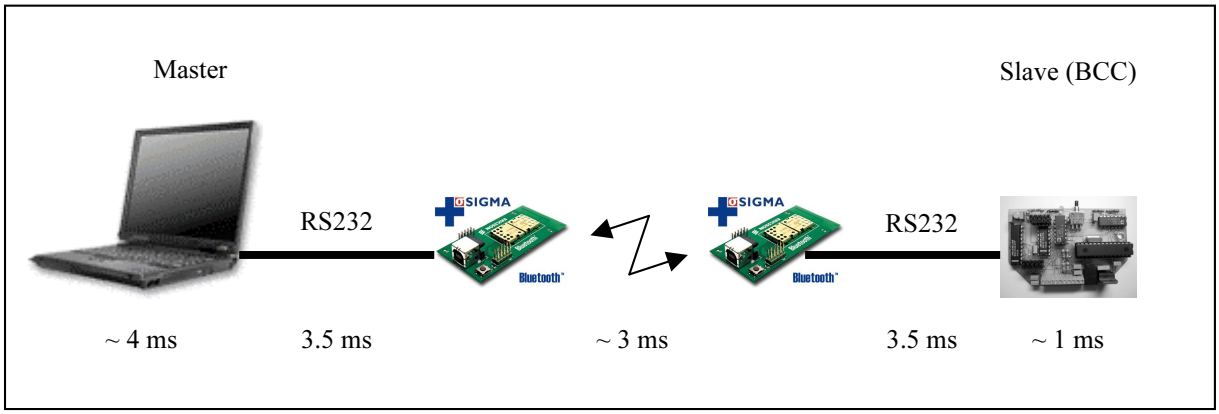


Figure 1: The Figure shows the measured delay time in the system. A data package is delayed approximated 15 milliseconds in both the slave to master direction and master to slave direction. This time is divided with 7 milliseconds on the RS232 serial ports, 3 milliseconds in the air, 1 millisecond in the slave software and 4 milliseconds in the master software. The time on the serial port has been measured and is according to the theoretical value. The time spent in the air is almost a factor 2 larger then the theoretical value, which can not be explained. Time spent in the software is only estimated.

Figure 2 shows the data package. The theoretical time for the data package to propagate on the RS232 serial ports, with the speed of 56.7 kbps, is 3.5 milliseconds. The package is 20 bytes long, 200 bits including start and stop bits. This time has been measured and found to be correct.

The speed of the Bluetooth connection, using DM1 packages, is in both directions 108.8 kbps, see Appendix F Section 2. The theoretical time to propagate over the air with this speed, including the error coding bits, is 1.8 milliseconds. Measurements have been done to investigate this and the time found was around 3 milliseconds. This is almost a factor 2 larger then the theoretical value, which can not be explained.

Combining the theoretical and measured value gives a delay in the software of the master and slave between 5-6 ms. Most of this time is probably derived from the master where the software is threaded and written in Java.

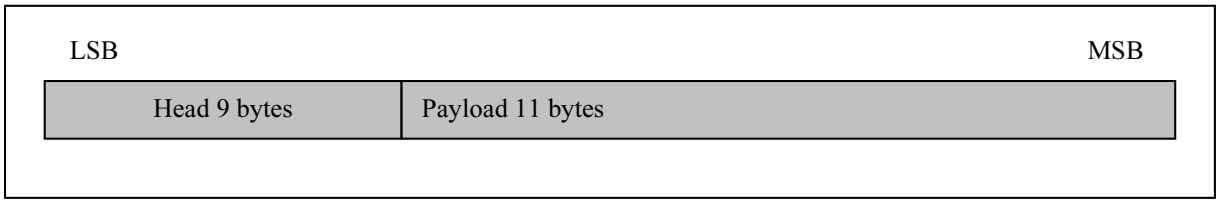
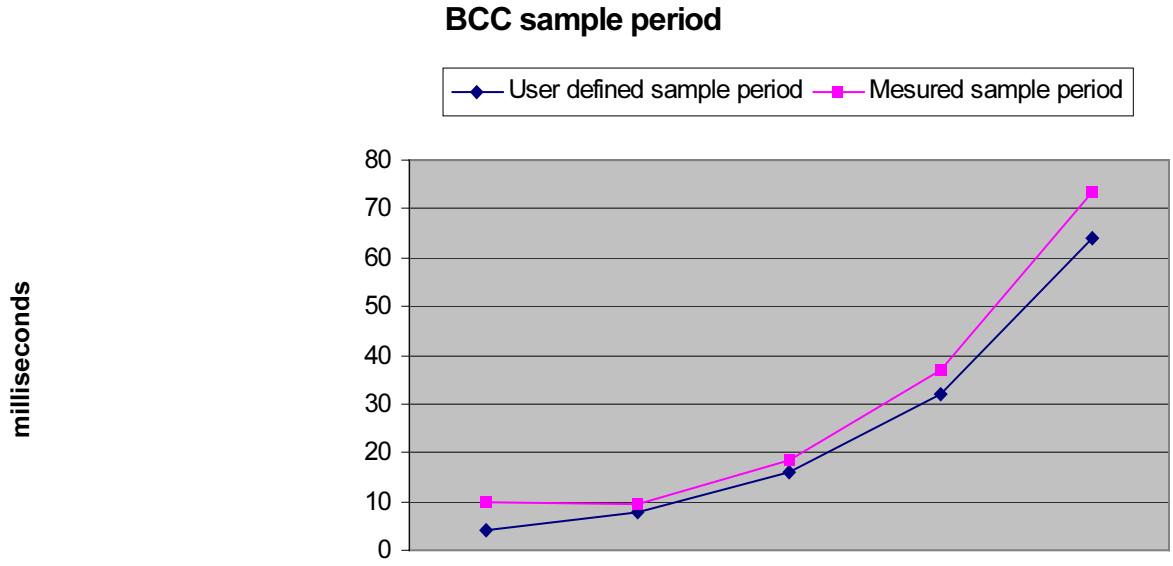


Figure 4.1: The Figure shows the general structure of a master to slave or slave to master data package. The head is illustrated in Figure 4.2 and contains Bluetooth specific information in order to propagate on the Bluetooth channel. All data is contained in the payload field and can be seen in Figure 4.5 and 4.8.

The minimal input sample period in the slave is constrained to 6 milliseconds. Measurements on the system have shown that this rate is the fastest achieved, with the functionality of the BCC intact. This is shown in Figure 3.

The software in the BCC is interrupt driven, which impose that the actual sample period always will be a bit higher then the user set time. Lowering the interrupt overhead in the system will allow an actual sample rate faster then 6 milliseconds. The software in BCC interrupts each 256 microseconds. To boost the performance the interrupt rate should have been each millisecond. This has not been done due to time constraint in the master thesis project, and the actual impact it would have on the performance is hard to estimate.



	1	2	3	4	5
User defined sample period	4	8	16	32	64
Mesured sample period	9.8	9.5	18.6	37	73.5

Figure 3: The Figure show the difference between the user defined sample period, and the measured sample period presented to the API in the master. The micro controller is interrupt driven which mean that the measured sample period always will be a little bit larger then the user defined. When the user defined sample period is set to 4 milliseconds the overhead imposed by the interrupts are so large that the measured sampled period is constrained. The maximal sampled period is achieved when the user defined sample period is 6 milliseconds.

The maximal simultaneious speed of master to slave and slave to master communication has not been mesured. This issue is discussed in Section 4.1.

The Bluetooth connection, that provides the BCC with wireless communication functionality, performs retransmission on lost packages. The retransmission time can, according to the Bluetooth specification 1.0 B [2], be user defined. This is not supported by the Bluetooth firmware, which continues the retransmission until it succeeds transmitting the package. This is not a good control strategy, as the Bluetooth module might continue to retransmit an old package even if a newer package is available. Our intuitive retransmission control strategy would have been to retransmit a package until the next package becomes available.

Appendix I: User's manual for the Bluetooth controlled Beetle

Before the program is started the Bluetooth modules need to be plugged in, and the car and the joystick should be supplied with power. All the devices should also be reset. After the completing the following actions the Beetle can be steered by the joystick:

1. Press the two buttons that says "Init". The Bluetooth modules are then initialized. Check the fields next to the buttons to see whether the command succeeded or not.
2. To search for Bluetooth devices in the proximity press the button "Search for Bluetooth devices". It takes about five seconds until the Bluetooth modules are finished with the inquiry. The devices that are found are then displayed in the field under the text "Available BT-devices". Hopefully both the joystick and the car are found. In that case the texts "Joystick" and "Radio Control Car" will be displayed in the column "Device". If unknown Bluetooth devices are found they are listed as a "Unknown device".
3. Press the combo box in the "Command" column to connect the car and the joystick. The commands "Connect" and "Disconnect" appear when the combo box is clicked. Choose "Connect". In the column "Status" the success of the command is displayed. Sometimes the command fails and it has to be given again.
4. If it is desired to limit the speed this can be done with the speed slider down to the right in the GUI.
5. Press the button "Steer with joystick" to steer the car.

Distance control was not implemented so the distance slider does not have any function. It is best to exit the program by pressing the cross in the top right corner. Signals to reset the Bluetooth modules are then sent and the program can usually be restarted without pressing their reset buttons. If something goes wrong when the connections are created it is recommended to open the debug window to see the information. Go to the menu "View" and press either "Errors" or "Errors and info". Error messages begin with "E ->" and information messages begin with "I ->".