KUNGL TEKNISKA HÖGSKOLAN

Building a platform for Bluetooth development with focus on the X-Mouse

Bachelor Thesis
Royal Institute of Technology, Kista

Niklas Agevik
nickesolna@yahoo.se

Stockholm, Summer 2002

Abstract

Our goal was to improve the intelligence of Aesthetically Pleasing Electronics (APE) product X-Mouse by implementing a new solution for Bluetooth in its next generation prototype. A prototype was already available which implemented a crude way of sending data via Bluetooth which worked fine for a prototype, but was not enough to show how the mouse seamlessly could be moved from PC to PC (one of the key benefits of a Bluetooth mouse). When researching the subject it soon became clear that APE needed a platform for using Bluetooth in the X-Mouse and their other upcoming products. By learning about Bluetooth we realized that the X-Mouse needed to implement a concept named “profiles” which is a part of the Bluetooth specification.

APE also requested that the solution should be flexible and cost-effective, allowing it to be used the in other upcoming prototypes.

We successfully identified and secured all the parts needed for a complete embedded Bluetooth system and negotiated with all mayor Bluetooth stack producers to ensure the best possible solution. IAR Systems AB was able to let us use their stack for evaluation and we could adapt it to suit our hardware platform and communicate with a Bluetooth module.

The embedded Bluetooth system we built is very flexible and is a fully certified solution which meets all demands that APE had. A roadmap has also been created for APE on how to proceed to implement it in their X-Mouse and other products

Acknowledgements

I have the following people to thank for the completion of this report (in no particular order):

- Johan Lundahl (Colleague, APE)
- Anders Graffman (CEO, APE)
- Jonas “Bigboy” Wikström (CTO, APE)
- Anders “Chadden” Hagberg (CFO, APE)
- Anders Västberg (Supervisor, KTH)
- Anders Wahlström (Developer, Lundinova)
- Marcus Pehrsson (Developer, Lundinova)
- Magnus Gustavsson (Broadband Technology)
- Lars Sjöberg (Sales & Marketing Manager Bluetooth, IAR Systems)
- Paul Kanarbik (Product Marketing Mananger, ACTE Components)
- Linus Norrbom (Customer product manager, Blue2Space)
- Mats Karlsson (CTO, Blue2Space)

Table of contents

1 Introduction

Aesthetically Pleasing Electronics (APE) was founded in 2001 around the concept of the X-Mouse, which is a crossover between a regular mouse and a joypad. I.e. it can be used both for leisure in the living room or as a normal computer mouse. It is also a remedy against Repeated Stress Injury (RSI)[1]. Together with APE's technological partner Lundinova a prototype was created which was later demoed at Comdex fall 2001 together with Toshiba.

The main objective of this thesis was to develop a more sophisticated prototype of a Bluetooth platform for use in an upcoming prototype of the X-Mouse. This prototype should primarily be capable connecting to any Bluetooth-enabled computer running any flavor of the Windows operating system. The platform built should also be flexible and adaptable allowing it to be used in other prototypes by APE.

The main focus of this report has been to investigate how to build a platform for developing cost-effective embedded Bluetooth for use in a consumer product. Integrating Bluetooth can be done in many different ways and some pointers for possible solutions are given in this report.

This report should be interesting to anyone that currently is looking for a cost-effective solution to implement Bluetooth either in their existing systems or in a new product.

1.1 Prequsities

The reader should also have a basic understanding of the current technologies used today for datacommunications. Internet-technologies and an understanding about radiocommuncations can also be helpful.

[1] "Musarm" in swedish

2 Description of Bluetooth

To understand this report fully an understanding of the Bluetooth technology is required.

The Bluetooth standard developed by the Bluetooth SIG [2]encompasses many layers and protocols. It is not necessary to know about all the layers to implement Bluetooth in a product and this part will focus on the layers needed for the actual implementation of Bluetooth and only give a shallow understanding of the layers which a user building an application never access directly.

2.1 About Bluetooth

Initially, when Bluetooth was conceived it was regarded as a simple cable replacement protocol. Nowadays the Bluetooth community has broadened its goals and the technology is set to be the de-facto standard for ad-hoc wireless connectivity. Still, on a side note, cable elimination is an important part of Bluetooth.

The standard was initially founded by Ericsson AB in 1994 and now consists of well over two thousand companies that are part of the Bluetooth SIG. The companies who since December 1999 are the core of the Bluetooth SIG Inc. are: 3Com, Agere, Ericsson Technology Licensing AB, IBM Corporation, Intel Corporation, Microsoft Corporation, Motorola Inc., Nokia and Toshiba Corporation.

Bluetooth SIG itself states its goal as:
"The Bluetooth™ wireless technology is set to revolutionize the personal connectivity market by providing freedom from wired connections. It is a specification for a small-form factor, low-cost radio solution providing links between mobile computers, mobile phones and other portable handheld devices, and connectivity to the Internet. The Bluetooth Special Interest Group (SIG), comprised of leaders in the telecommunications, computing, and network industries, is driving development of the technology and bringing it to market."

[2] Bluetooth Special Interest Group, http://www.bluetooth.org

Bluetooth aims to be a “smart” technology. That is, as two Bluetooth-enabled devices approach each other they can automatically make contact and exchange what services they provide. A standard set of services, or more commonly referred to as profiles, has been set and more are to come. A profile is a standard for how two devices operating over Bluetooth can access each others functions. Some profiles are discussed in depth further in this report.

The benefits of this concept for the end user are evident. If the user can use an arbitrary headset together with his mobile phone and don’t have to care about connecting cables it can save him time and money. Different scenarios are discussed later in this report.

Bluetooth operates in the 2.4 GHz band, often referred to as part of the ISM (Industrial, scientific and medical band). The ISM is globally available and license free, therefore it can be used without permission from a regulatory agency such as Swedish PTS (Post och Telestyrelsen) or the US FCC (Federal Communications Commission).

2.1.1 Bluetooth stack layers

A common way of modeling complex systems with many functions is to divide them into modules that are linked together. In the OSI reference model [3]these modules are named layers and the same naming scheme have been adapted for Bluetooth hardware and software. The idea is that each layer performs a special set of functions, only working with the layer directly above and below itself. I.e. a layer that is positioned high in the stack does not need to be aware of how the actual radio transmissions are performed.

The Bluetooth standard has so far reached version 1.1 and for a Bluetooth application to function all layers of the stack has to comply with the same version of the specification. Several interoperability problems were discovered when stacks and hardware from different manufacturers were used that implemented version 1.0b. The 1.1 version is set to resolve these issues.

[3] A way of modeling any networked system. More information at: http://www.cs.bsu.edu/~peb/cs637/layering/ISO.htm

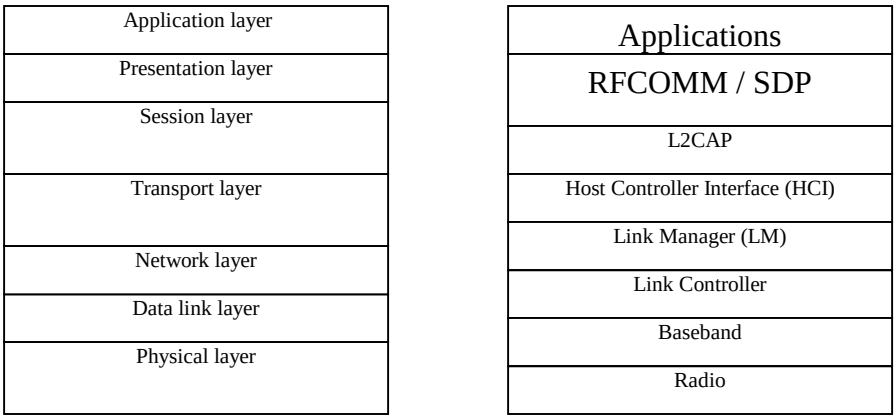


Figure 1: The OSI Reference model and the Bluetooth stack model

2.1.2 Radio and baseband

The baseband and radio are the lowest layers of the Bluetooth stack model and in a comparison to the OSI reference model this is equivalent to the physical layer.

As already stated Bluetooth operates in the 2.4 GHz frequency band and three different power classes have been defined for using Bluetooth over different ranges.

- Class 1: 100m (100mW)
- Class 2: 20m (2.5mW)
- Class 3: 10m (1mW)

By using a FHSS [4]technique Bluetooth devices can avoid most of the interference from other devices and enable higher data-rates. Each timeslot lasts for 625 µs and one packet will be one, three or five timeslots. The maximum theoretical data rate when sending packets consisting of five timeslots is 723 kb/s. If both devices are be sending at their maximum data rate their respective speed is 433.9 kb/s. This speed slowly degrades when moving up in the Bluetooth stack since all layers add some overhead.

[4] Frequency hopping spread spectrum

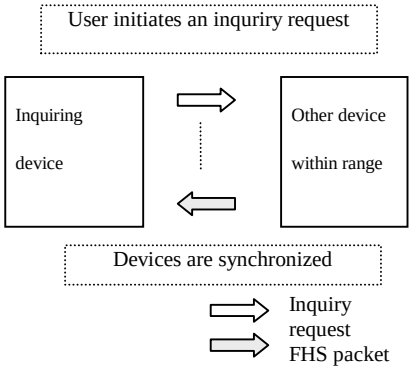


Figure 2: Synchronization sequence of two Bluetooth devices

After sending a complete packet the Bluetooth device makes a frequency hop[5]. This is to ensure that a packet lost due to interference is resent on a different frequency. During normal operation the device performs about 1,600 hops per second. When two devices come within range one of the devices can search for neighboring devices and eventually initiate a connection. For this to be possible the inquiring device has to synchronize its hopping sequence to the other device(s). The device initiating the inquiry sends out a series of Inquiry packets. Eventually a responding device will answer with a *Frequency Hop Sequence* (FHS) packet. The FHS packet contains information that allows the two devices to synchronize themselves and initiate a connection. This process is shown in Figure 2 .

When the FHS packet is successfully received one device can initiate a connection and one of the devices are selected as a master, creating a point-to-point connection. If more than one device is connected to the master he becomes the host of a "piconet". Up to seven devices can communicate simultaneously with the master. When this happens the master allocates bandwidth, and according to the different devices data requirements, a number of timeslots are allocated to each device.

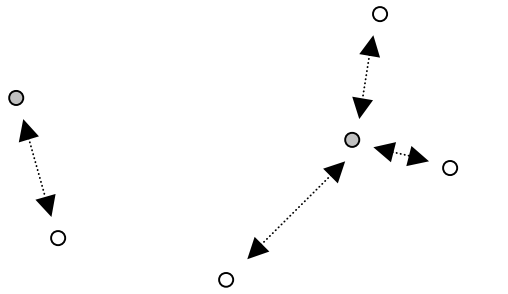


Figure 3: A point-to-point and a point-to-multipoint connection

All devices carry a unique identifier consisting of a 48-bit IEEE Mac Address called "Bluetooth Device Address". It is commonly used so that applications can remember which devices they have talked to before and apply such things as security settings for each individual device.

[5] Change of frequency

2.1.3 HCI

The HCI (Host Controller Interface) is a standardized interface between a Bluetooth module and the higher layers of the stack. A product developer can select any combination of a Bluetooth module and stack to control the lower layers of the stack. HCI commands can be sent over a number of different channels including RS232, PCMCIA and USB. The lower layers of stack, which are implemented in a mix of hardware and software, are often referred to as firmware while the higher layers are referred to as the stack.

When using Bluetooth on a computer this means that the Bluetooth hardware is attached to the computer via any of the supported channels (i.e. a USB-port) and everything above the HCI-layer is run on the computer.

Bluetooth can of course also be used in embedded systems like mobile phones, mice etc. When this is the case the stack is run on an embedded processor which controls the Bluetooth hardware via HCI.

2.1.4 Physical links: SCO and ACL

The Bluetooth standard offers two ways of sending data. ACL (Asynchronous Connection-Less) links are the basic links and are established as soon as two devices within range are connected. It always exists none or exactly one ACL link between the master and a slave in vicinity. Broadcast packets are always transmitted via ACL.

The ACL link does not ensure that data will be transmitted in a regular fashion to a slave. Data will be sent sporadically when a slot and data is available from the higher levels of the stack. An ACL-link can be configured so that lost packets are resent. ACL can be regarded as a packet-switched connection.

The SCO (Synchronous Connection Oriented) link differs in many areas from ACL. When a SCO link is used the master and slave uses a reserved channel bandwidth and a periodic exchange of data in the form of reserved slots.

SCO is used for time-bounded information like video, audio etc. Since SCO packets are time-bounded, lost packets are never retransmitted. A slave can use up to three SCO-links from the same master.

2.1.5 L2CAP

The Logical Link Control and Adaption Protocol (L2CAP) is used by higher levels of the stack to send and retrieve data in a simple fashion. It should be noted that it only functions on ACL channels. Among the things L2CAP does is:

- Multiplex packets originating from higher level protocols letting them share a single ACL link
- Segmentation of long packets and reassembly of packets
- Quality of service management (QoS)

L2CAP is the standard way of transferring ACL data from higher level protocols because it frees the application from such things as packet segmentation. It also contains a Maximum Transmission Unit which can be set by higher level protocols.

2.1.6 RFCOMM

The legacy way of transferring data is via RS-232 which has nine pins used for signaling and data (see Figure 7). The RFCOMM layer can emulate all these pins and also support different port settings (baud rate, parity etc). It is built upon L2CAP and uses it for transferring data.

There are two types of RFCOMM entities:

- Type 1 – Internal emulated serial port
- Type 2 – Intermediate device with physical serial port

Type 1 is usually a virtual serial port that is created by an SPP [6] (see below) server to look like a regular COM-port on a PC. Type 2 is used where a legacy device is connected to a Bluetooth device which acts a cable stand-in.

[6] Serial Ports Profile

2.1.7 Profiles

Bluetooth has the means of carrying data across the air and all Bluetooth devices are able to send and receive data. The Bluetooth SIG could have stopped here by providing a simple means for proprietary applications to exchange data over a standardized interface, but they didn't.

A concept named profiles helps devices request another device to perform a function for them.

2.1.7.1 Basic profiles

The most important profile, which is the base for all other profiles, is the *Generic Access Profile* or GAP. Its purpose is to let all devices establish a baseband-link by defining a set of discoverability modes that will allow devices to synchronize and exchange data.

Also important is the *Service Discovery protocol (SDP)* which should not be mistaken for a profile. This protocol provides functions for storing and accessing a list of services that a device offers. A device that only functions as a client does not need to implement this protocol.

The *Service Discovery Application Profile (SDAP)* is the counterpart of SDP. It provides a way for two devices that have established a connection to access their respective SDP records.

2.1.7.2 Other profiles

Building upon this profile a total of 13 profiles are currently available from the Bluetooth SIG and more are to come. By using profiles, equipment from different manufacturers can interact and easily exchange data. I.e. an Ericsson headset interacts with a Nokia mobile phone or in the case of the X-Mouse, a mouse interacting with an arbitrary host capable of using a HID.

A full list of all profiles is available in appendix 1.

As time progresses more ideas for profiles will come up and be added to the standard set of profiles. A company can also develop a proprietary profile if their application requires a very special set of commands.

2.2 Security

Since anyone within range could potentially monitor transmissions over air security is important. If data is sent over air it must be

1. Not in any way harmful to interception
   or
2. Encrypted or in some way distorted so that a monitoring party cannot gain any useful knowledge about the transmission

Bluetooth applications use the first alternative when appropriate but endorse the second alternative for most transmissions. For most profiles encryption is optional. Most applications, like a Bluetooth keyboard should use encryption since passwords or other sensitive information may easily be stolen by a third party.

The SIG has adopted the SAFER+ cipher algorithm[7] to authenticate devices and encrypt transmissions. An encryption stream is generated based which is then exclusive-OR'd (XOR'd) with either the data being transmitted or received. The encryption stream is based on the following information:

- The master's Bluetooth device address
- The master's Bluetooth slot clock
- A secret key (PIN) shared by both devices in a point-to-point connection or shared by all devices in a piconet

The secret key can of course not be shared by transmitting it over an insecure Bluetooth connection. For a keyboard the screen can display a PIN-number which is then entered on the keyboard. In a wireless mouse scenario where input options are limited the mouse can be supplied with a PIN-number which is then entered at the host. A PIN can according to Bluetooth specifications be up to 128 bits (16 bytes).

When two devices "know" that they are sharing a link and are sharing a common link key this is referred to as they are *bonded*.

[7] NIST (National Institute of Standards and Technology) Web site, http://www.nist.gov/aes

2.3 Applications

There are several entry points in the Bluetooth stack where application developers can work. An application that only requires data to be sent in a point-to-point or a point-to-multipoint setup can work directly with the HCI layer. In this case the developer needs to explicitly perform all necessary functions on both ends of the connection. This technique is useful where Bluetooth only will be used for a very limited function and interoperability and ad-hoc connectivity is not a concern.

In these cases Bluetooth can be regarded as a low-cost way of transmitting data. Bluetooth modules are predicted to reach high volumes which lower the price for a module to only a few dollars[8]. Knowledge and technical expertise are also widely available from consulting firms. This approach would be more cost-effective than developing a proprietary system.

Example:
A manufacturer uses robots in an industrial environment for production. Cables are used to connect the robots to a computer that controls them. The cables are wearied down quickly and soon become a high cost for the company.

Here Bluetooth can be used instead of the cables. Interoperability is not a concern since the application developed will only be used at the company where they can control both ends of the link.
Other devices in range will be able to detect the Bluetooth device but cannot access its data the proprietary software.

All other applications that require interoperability between different platforms and devices should position them as high as possible in the stack, preferably implementing both GAP and SDP. The device should also use a profile for its function.

Example:

[8] Elektroniktidningen 30/5 2002 , ”Bluetoothkrets för under fyra dollar” http://www.elektroniktidningen.se/uploaded/template/asp/eltmall.asp?version=6439

A Bluetooth-enabled PDA[9] comes in range with a computer running a mail-client. The two devices, both implementing the “Synchronize”-profile recognizes each other and mail that is written on the PDA is sent to the computer and newly arrived mails are delivered to the PDA.

If a developer is creating a mail client for a PDA, implementing the “Synchronize”-profile will be sufficient to communicate with a wide range of mail clients from different manufacturers.

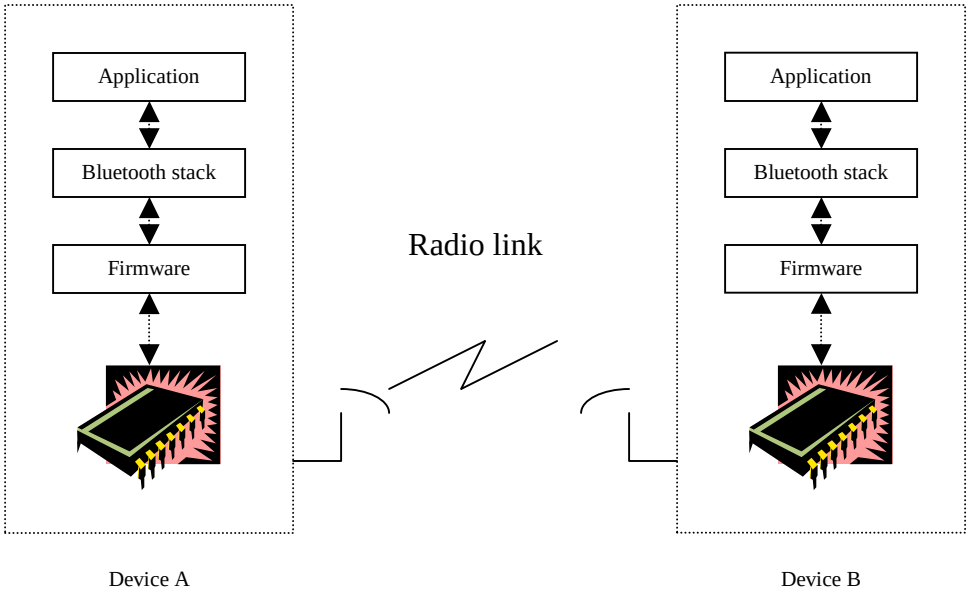


Figure 4: Two devices communicating over a Bluetooth link

2.4 Other related techniques

A number of other technologies have been discussed as competitors to Bluetooth. These are different kinds of Wireless LAN technologies (WLAN). These technologies are also wireless technologies which enable an ad-hoc connectivity. But they are not designed for

[9] Personal Digital Assistant, handheld computer. I.e. a Palm Pilot.

small personal networks where devices quickly disconnect or enter the network. Their primary focus is on high data-rates and mobile access. This is not the case for Bluetooth.

The prevailing opinion in the industry which I agree with is that Bluetooth and WLAN will complement each other. It is already possible for developers to build circuits that incorporate the two technologies[10].

[10] Mobilian, "Enabling Wi-Fi™ and Bluetooth™" – "Coexistence without compromise", http://www.mobilian.com/TrueRadio_frame.htm, available as of January 2002

3 Method

The goals of the project were clearly set at the beginning. Our roadmap for getting there was however undefined and our first task was to complete a roadmap for the project. Since we at the beginning of the project had no idea of how the current prototype was implemented, the first step was to gain as much knowledge as possible from Lundinova about their prototype and their ideas on how to proceed.

3.1 First prototype

The solution used in the prototype by Lundinova was not profile dependent. Instead it used a custom solution that only used Bluetooth for sending raw data serial data to a dongle attached to the host.

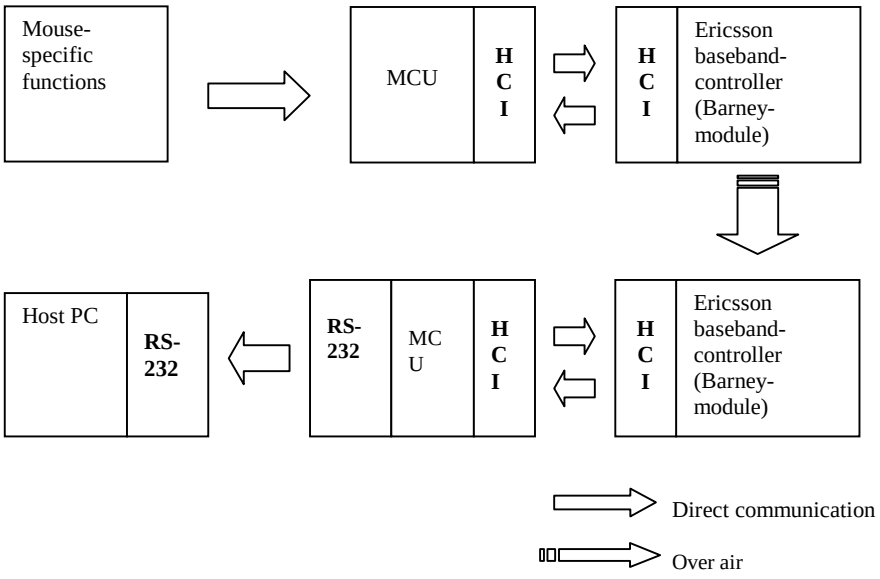


Figure 5: The initial prototype as supplied by Lundinova

This is a good solution of a complex problem and works satisfactory for a prototype. So why not stick with it? First of all it occupies one of the host's serial ports[11]. Since one of the benefits of using a Bluetooth-enabled mouse is that it keeps your physical serial ports

[11] In this case a COMx port where x stands for the port number.

open for other devices, the Bluetooth solution looses one of its benefits. It also ties the user to a proprietary solution which is incompatible with other computers running Bluetooth.

3.2 Roadmap

After gaining knowledge about the Lundinova prototype we could start working on our first roadmap which is the one presented below. The roadmap evolved during the first two weeks of the project with only slight modifications later on. We decided that a form of "glue-logic" would be used where we attach our MCU[12] somewhere in-between the current mouse-controller and the baseband-module.

1. Investigate what profile(s) should be used or if one should be developed.
2. Select a cost-effective platform for implementing the profile(s) consisting of a baseband-processor and MCU (Microcontroller Unit).
3. Implement selected profiles on selected platform.
4. Perform interoperability tests with different Bluetooth modules/platforms.
5. Move the implementation into the actual X-Mouse.
6. Prepare anything that can help the industrial partner for mass production.
7. If possible, investigate any power saving techniques for the mouse. Several have already been discussed internally at APE.

On a separate track we had to find which mouse protocol we would be using for the X-Mouse data stream. We wanted one that would be compatible with all mayor operating systems and was a de-facto standard in the industry.

When we started to look at ways to complete the different steps we concluded that there were many, many ways to go with no company offering a complete solution. We tried to wait as long as possible before locking ourselves into a solution. Our focus was also on creating a solution that would not only be viable now in a prototype but that could also be used in the future if the X-Mouse would enter mass-production.

[12] Microcontroller Unit, a microprocessor optimized to be used to control electronic equipment.

This proved to be a difficult task. Most solutions we found included companies working closely together with us. After discussions with APE we concluded that this would be way too expensive and it was also our intention to build as much as possible ourselves and gain intellectual property for APE.

3.2.1 Selecting a profile

We decided early on that a profile should be implemented in the mouse to ensure interoperability and create a no-hassles solution for the end user.

3.2.1.1 HID-profile

The Bluetooth SIG has created a working group for developing a profile named "Human Interface Devices" (HID) which was developed with mice and keyboards in mind. It gives us the following primary advantages:

- A special boot-protocol. Enables the mouse to be used before a operating system has booted up. An example of when this would be necessary is when using a BIOS or other simple tool requiring a low amount of memory.
- Easy plug-and-play solution for customer. Just place the mouse nearby a computer which it is bonded with and it will start to function.
- No need to develop a proprietary solution for power-saving mode or drivers for each operating system.

The best solution for the X-mouse would of course be to use this profile but when we researched the subject a couple of problems became evident.

- Currently the profile is not finished and is subject to change. It is currently in version 0.95 and is expected to be finalized during Q3 2002.
- No manufacturers of embedded stacks has of yet implemented the profile even though everyone we held discussions with had it in their roadmap.
- No operating system supports the HID-profile. This means that we would have to develop the host-side profile ourselves. Both Apple [13] and Microsoft [14]have it in their roadmaps though.

[13] Apple press release, http://www.apple.com/pr/library/2002/mar/20bluetooth.html, available as of March 2002

[14] Microsoft press release, http://www.microsoft.com/hwdev/tech/network/bluetooth/default.asp, available as of May 2002

The solution to this problem would of course be to develop both the client and host side software ourselves but as we researched this possibility it became evident that this was not feasible in our relative short timeframe. It would also be very expensive as we would have to certify it with a BQB [15]to ensure interoperability and be able to use it in production. Lars Sjöberg at IAR estimated that their stack was the work of many man-years.

Still, using the HID-profile would be the best way to go and our research shows that as soon as it is complete it should be integrated in the X-mouse because of the advantages gained.

3.2.1.2 Serial Port Profile

The second profile which was of importance to us was the SPP (Serial port profile). The SPP gives the computer a virtual COM-port, just like the ones (usually two) already present on all PC's and laptops. The SPP has several advantages:

- Works with legacy operating systems. Even if the operating system doesn't support Bluetooth natively most Bluetooth enablers (dongles, plug-in cards etc) supports the SPP.
- Easy to implement since it is already available for all stacks we have studied.
- About all operating systems to date comes with a driver for serial mice.

The disadvantages being that it lacks native support for any advantages that could be gained from the HID-profile (see earlier paragraph). If we would reach a point later, where the HID-profile becomes widely available, the SPP part can still be contained in the mouse, supporting older operating systems.

Last, APE had a special request for functions that would be specific to their mouse. Their requests were neither supported by the HID-profile nor was it feasible to implement them using the SPP. Our solution to this was to suggest that a proprietary profile should be developed with support for APE-specific functions.

---

[15] Bluetooth Qualification Body, a standard set by the Bluetooth SIG for qualification of BT-devices

3.2.1.3 Conclusions on profile selection

For the reasons presented we decided to use the following roadmap for profile-selection:

1. Find suitable stack with SPP support and a roadmap for implementing the HID-profile.
2. Implement stack with support for only SPP.
3. As soon as the HID-profile is complete implement it together with the SPP.
4. Build proprietary profiles for X-Mouse specific functions

3.3 Selecting a platform for the Bluetooth stack

3.3.1 Requirements

We had several requirements for our platform. First of all we needed a cheap solution since the X-Mouse is a consumer product. Also, we needed one that wouldn't prove too difficult for us to develop on. It should be possible to perform our task at hand within the limited timeframe and scalable enough to accommodate needs that may arise in the future.

If possible and feasible we wanted one that mentors at Lundinova had knowledge about since we realized we would need help in the process of integrating it with the stack.

3.3.2 Options

Since we had identified a need to use all layers in the Bluetooth stack all the way up to the profiles level we needed to select a processor that had the capacity to handle these functions. Lundinova recommended a solution from Atmel. After interviews and talking to Atmel[16] we concluded that this would be a good solution since a good part of all Bluetooth devices are running on a microcontroller (MCU) from them. It was also a common notion from all stack manufacturers we held discussions with that Atmel had a cost-effective solution. Because of our limited timeframe we decided not to research other viable platforms.

Atmel's two primary platforms of interest for us was the "AT91 ARM Thumb" and "AVR 8-bit RISC". These are both 8-bit platforms with the ARM being the more powerful of the two. Our research showed that several Bluetooth devices currently on the market were using the ARM[17]. Blue2space told us in an interview that they had selected the ARM because of their technical expertise in this platform but they believed that the AVR would also be viable for running a lightweight version of a stack.

Another option we looked into was the possibility of using a solution from CSR[18]. CSR is one of the few companies that offer a complete and cheap solution (compared to other

[16] Developer of MCUs (among other things), http://www.atmel.com
[17] After talking to Blue2Space and internal information from Atmel
[18] Cambridge Silicon Radio, http://www.csr.com

complete solutions) for developing Bluetooth products. APE did not grant us the financial resources to invest in their development kit and we were unable to borrow or otherwise obtain one. Despite this, based on information we received from CSR, we concluded that the CSR-BlueCore technology could be a good solution. Based on the fact that we were unsure of it abilities to run a certified version of SPP we decided not to follow up on this. But it should be noted that CSR has many interesting solutions on their roadmap that very well could prove valuable to APE in the future, specially the BlueCore2-HID[19].

It was more or less impossible for us to reach a decision since we had no idea what amount of processing power a complete Bluetooth system would require. Finally we chose the AVR because of its easy-of-use and the knowledge that Lundinova had it.

[19] A chip dedicated to support Bluetooth HID-devices.

3.4 Bluetooth hardware and software

3.4.1 The case for the stack

A Bluetooth stack is needed to perform all high-level functions. Usually the stack consists of the bottom layers implemented in hardware and software (firmware) and another high-level stack accessing these functions. These two can be from different manufacturers and use a standardized interface to ensure interoperability. It should be noted that the interoperability between the hardware modules and the higher-level stacks is not 100% because of mishaps during the implementation of stacks/hardware by different manufacturers.

3.4.2 Selecting Bluetooth hardware

Selecting Bluetooth hardware is of course crucial to make a low-cost product but it is outside the scope of this report. We decided to stick with Ericsson Bluetooth modules because of the availability and the fact that the X-Mouse already had such a module. Initially we bought a development board including an Ericsson module but many companies were kind enough to lend us other (more advanced) Ericsson boards. Another reason was because of the fact that Ericsson modules are known to be interoperable. Even though all Bluetooth hardware should be interoperable this is not always the case.

Even though Ericsson modules were used during this project it is possibly not the best choice for a consumer product.

3.4.3 Creating a stack

One option for us which we looked into was to build a stack ourselves. We tested this idea with our mentors but it did not seem like a viable solution. This would also ruin any chances for us to build a certified solution in anything less than a year. Another thing that stopped us was the fact that even if we could implement a working 1.1 version of the stack it would not take long before the stack would have to be upgraded to support a new version (primarily 2.0 which is on the Bluetooth SIG roadmap[20]).

[20] See http://www.bluetooth.org

3.4.4 Obtaining a stack

A number of companies currently offer stacks and we held discussions with most of them. One of our priorities for the stack was that we should be able to work closely with it and gain in-depth knowledge about it. It should also be cost-effective and the company offering it should be a stable one with a roadmap for implementing the HID-profile and solid help if interoperability should prove to be a problem.

We also wanted to run it on an AVR as stated above. Of course we had to be flexible on this but we were assured that it would be possible. The following companies offered stacks that we looked at:

- Mecel
- Bluetools Online
- Atinav
- Comtec Telca (Ericsson reference stack)
- Axis
- IVT Corporation
- Stollman
- Widcomm
- Extended systems
- Microsoft (no embedded edition)
- NoHau (C-Blue)
- IAR Systems

We based the evaluation upon product descriptions, interviews with the manufacturers / resellers and the evaluation packages for Win32 systems that came with most stacks. After seeing different evaluation packages we concluded that the stack configuration should not be taken lightly and must be taken into account when selecting a stack. Since we wanted to configure the stack ourselves we also added this to our list of demands.

The price range was from 100 KSEK to 300 KSEK (11 200 € - 33 600 €) and many of the companies did not offer any prototype licenses or evaluation licenses for embedded systems. One of the companies that could fulfill all of our requirements was IAR Systems

AB. Another positive aspect with IAR was the fact that their offices were close to us. After negotiations we acquired a copy of their stack and the configuration tool MakeApp.

3.4.5 Turning the device discoverable

As soon as the stack is running it has to instruct the Bluetooth module to be discoverable. The stack should handle this transparently by creating HCI packets, which are then sent to the Bluetooth module.

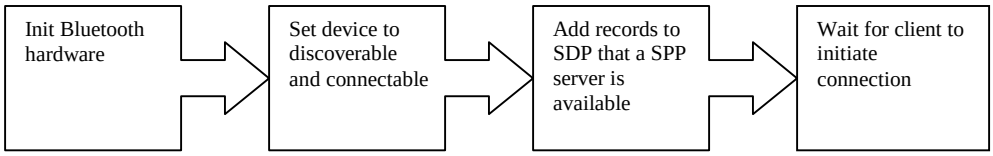


Figure 6: Setting up a SPP server

The next steps as outlined in Figure 6 are to set up an SDP server internally, within the stack that can handle requests via SDAP. When this is running it can be updated so that a connecting party can access its SPP server. If we were to implement the HID profile in the future, we would of course also add it is an entry to the SDP server.

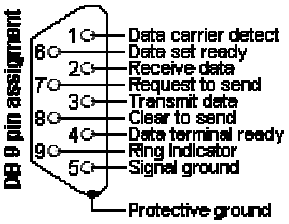
3.5 Mouse interfacing

3.5.1 Internals

We had to make an overview of which mouse-protocols were available and which would suit us. This research was mostly carried out on the Internet. We also looked at a standard Microsoft Windows distribution to see what standard drivers it was equipped with. The following are protocols we decided to look closer at:

- PS/2 Mouse protocol
- Microsoft 3-Button Intellimouse
- Mousesystems mouse
- Serial mouse protocol

We also had a mouse with a D-SUB connector (see below) that we could use for testing.



The D-SUB connector (RS-232)

6-pin Mini-DIN (PS/2):
1. Data
2. Not used
3. Ground
4. +5v
5. Clock
6. Not used

Figure 7: Two types of mouse sockets

Since it used the serial mouse protocol we decided to use this one to make things easy for us.

3.5.2 Connectors

There are currently a number of connectors that a mouse uses. The most common nowadays is the USB[21]-connector. It was not of interest to us since we had decided to use the Serial Ports Profile for our purposes.

Most legacy mice use the D-SUB connector while newer mice use the PS/2 and according protocol. It should be noted that the connector used does not need to correspond with the protocol used. I.e. a mouse interfacing via the 6-pin mini-din PS/2 connector does not necessarily need to use the equivalent protocol (though this is the case most of the time).

[21] Standardized plug for accessing peripheral devices

3.5.3 Listening to mouse traffic

Our mentors at Lundinova said that the easiest way of implementing a protocol is to listen to traffic by a device already implementing the protocol and then follow it step-by-step. We decided to do this and built a device for listening to the traffic between our serial mouse and a Microsoft Windows system. The listener we used looked like this:

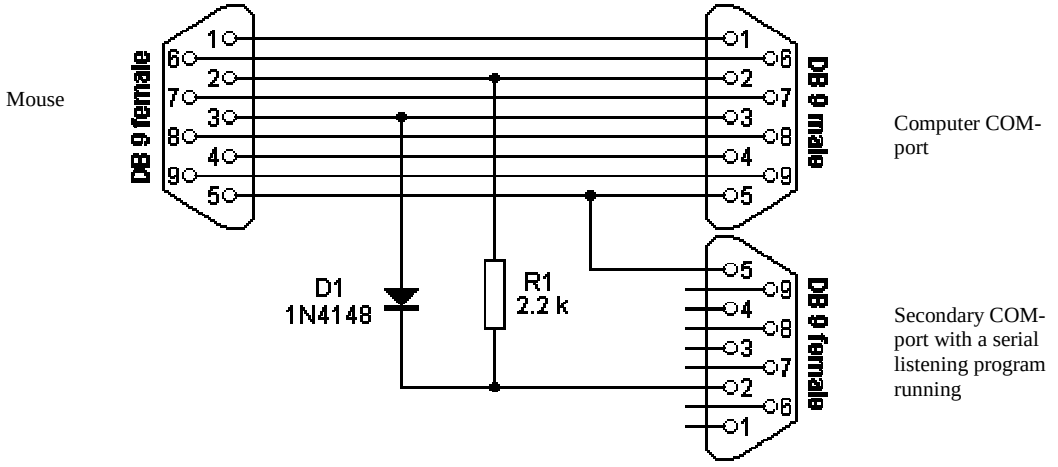


Figure 8: Device for sniffing an RS-232 connection

This setup also requires a secondary computer with a program running that can detect the traffic and present it onscreen to the user. We used a program named "SerialSniffer[22]".

3.5.4 Drivers

All modern operating systems are equipped with a driver for mice protocols to handle the data-stream originating from the mouse. We hoped that we would but able to use the supplied drivers for our solution. In the case that this wouldn't work, we ordered a Driver Development Kit from Microsoft so that we at least could get the mouse working on that operating system.

[22] Program available as of June 2002 at http://www.download.com

3.6 Development environment

Initially we hoped there would be a development environment available that suited our needs. We looked into a number of solutions from different manufacturers but the only one that seemed viable for us was the CSR Casira. The Casira comes with a complete development environment which can be used almost all the way on the road towards mass production.

Since CSR BlueCore solution was not an option for us (discussed elsewhere in this report) we had to setup our environment ourselves.

Of all the stacks we looked at all had a Win32 version which made it possible to develop most of the program logic using Microsoft Visual 6 C++[23]. For the embedded parts we selected an Atmel STK500 + 501 development board. ACTE was kind enough to provide us with engineering samples of one of the high-range Atmel MCUs.

Since the tool used for configuring the IAR stack generated ANSI-C source code, we also needed a compiler that could build this codebase into the "Intel-Extended" hex-format which is what the AVR uses.

The full stack was well above 25000 lines of C-code. To compile it we tried the following compilers:

- Codevision AVR (not fully ANSI-C compatible)
- Imagecraft for AVR
- IAR Systems Embedded Workbench AVR

IAR's compiler was the one that worked best and we selected this for our development. It was also the one with the best optimizing functions, which left us with more room on the MCU. It also came with a debugger (C-SPY), something that we valued.

Last, we needed a Bluetooth module that our program could communicate with. We used an Ericsson development board with a ROK 007 and ROK 008 module. Since the MCU

[23] C++ Compiler for the Microsoft Windows operating system

would be communicating with the Bluetooth module via an UART[24] we needed a MCU that was equipped with two UARTs so that one could be used for debugging. The ATMega128L which we had received from ACTE supported this.

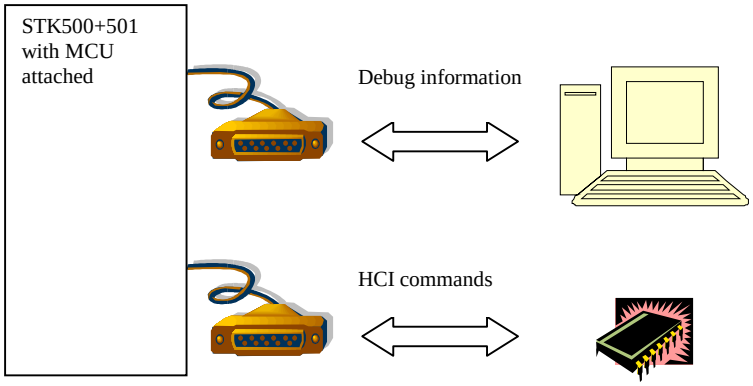


Figure 9: The development environment (see Figure 11 for a photo)

[24] Universal Asynchronous Receiver-Transmitter, used for sending and receiving serial data.

4 Results

4.1 Implementing a mouse

4.1.1 Emulating mouse-specific functions

The RS-232 listener that we had built worked like a charm. We could see all packets being sent between our mouse and Windows. But there came up a number of problems. First of all we were unsure with which baud rate/stop bits/parity the data was sent. We deduced this by trial-and-error and some searching on the Internet. The baud rate was 1200 baud with 8 data bits, one stop bit and no parity.

When we could listen to the transmission between the mouse and the computer it was very easy to match the data sent with one of the protocols that we had guessed that the mouse was using (serial mouse protocol).

	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Byte 1	Yov	Xov	Y8	X8	1	MB	RB	LB
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

Symbol definition:
LB: State of the left button, pressed = 1, released=0.
RB: State of the right button, pressed = 1, released=0.
MB: State of the middle button, pressed = 1, released=0 (if present).
Xov, Yov: Overflow. Indicates more than 9 bits of movement detected. X8-X0 or Y8-Y0 should be set to maximum magnitude in the appropriate direction.
X8-X0: 9-bit signed twos complement integer that presents the relative displacement of the device in the X direction since the last data transmission. A positive value indicates motion to the right; a negative value indicates motion to the left.
Y8-Y0: Same as X8-X0. A positive value indicates device motion upward; a negative value indicates motion downward.

Figure 10: The serial mouse protocol

We started to implement the mouse in our MCU but somehow Windows were unable to detect that we had a mouse attached. When we used our reference mouse it was detected

and the appropriate drivers were loaded. As soon as the operating system had booted we could disconnect the mouse and make a switch by inserting our development board with a MCU loaded with a program to emulate all mouse-functions. We programmed the buttons on our development board to perform different functions. I.e. with one button we could simulate a left mouse-click and with another we could move the mouse-pointer.

Soon we were able to see that there was a special power-up sequence that had to be followed for the computer BIOS and furthermore the operating system to detect the mouse.

When power was applied to our reference-mouse it dropped the RTS line for a few milliseconds which gave the computer the indication of a mouse being present. We decided not to try mimicking this condition with our development board. This decision was done because it would not gain the project anything since we were going to transmit the data over Bluetooth and no BIOS's to date support native Bluetooth.

Earlier we had believed that when the computer detects mice when it has loaded all Bluetooth functionality. Since this was not the case we could not expect to get X-Mouse to function over an emulated serial port on the host side before the operating system was completely loaded.

This problem cannot be solved before computers with native Bluetooth support (like native USB support) in BIOS become available. And this doesn't help our case since our motive behind implementing the SPP was to make it function on legacy computers.

Luckily enough this did not pose a big problem. We had to face the fact that we would not get the X-Mouse to function at boot-time but as soon as the operating system is loaded the user can instruct the operating system to search for mice and perform the same sequence as the one performed at boot time, this time with all Bluetooth functionality loaded.

4.1.2 Discovering via Bluetooth and SPP

By developing on the Win32 version of the IAR stack and running it against our Ericsson development board we could code most of the program logic using Microsoft Visual 6 C++ (MSVC6). This is a great advantage since it is a (relatively) long process to compile

and execute code on an embedded system. The debugging tools available with MCVC6 were also much more advanced than those that we had at our disposal for embedded development. The procedure used for making the Bluetooth-device discoverable is exactly the same regardless of what platform or stack it is utilizing.

With the IAR stack's API[25] we could easily perform the steps outlined in Figure 6. Since the IAR stack's inner workings is a trade secret of IAR Systems this report does not include any source code that utilizes the IAR stack.

When this was fulfilled we ran into a number of other problems. First of all we discovered that there were numerous incompability problems between Bluetooth modules from different manufacturers. We used the following adapters and modules:

- Xircom CreditCard Bluetooth Adapter (Xircom is a subsidiary of Intel)
- 3Com USB Bluetooth Dongle
- Ericsson ROK 007 with IAR Stack
- Ericsson mobile phone T39 equipped with Bluetooth

All of these had difficulties talking to each other. Sometimes we had problems accessing the SDP records via SDAP and we were at times also unable to retrieve the remote friendly name of the devices. We did however note that the Xircom card was using Bluetooth 1.0b, which we later upgraded to the 1.1 version of the standard. This did not solve all of the problems but we did get a higher rate of success.

The computer equipped with the Xircom card could now see and communicate with our computer running the IAR stack and Ericsson module.

We could now initiate any legacy program working over COM-ports and let it connect via a virtual COM-port to our program running on a Win32 system. This worked very well and we could exchange data just as if the two computers would be connected by cable.

Our next step was to let the operating system search for mice and see if it did sent any information on its virtual Bluetooth COM-ports. We were not able to receive any

[25] Application programming interface

information over Bluetooth when a scan for peripherals was executed in Microsoft Windows. Because of this we had to rethink our strategy. One way would be too write a new driver, something that we at that point did not have time to complete.

4.2 Running the Bluetooth stack

We began the two parallel tasks of porting the stack to the MCU and adding the program logic for the SDP, SPP and mouse-functions.

Our first attempts to run the stack were done on one of the low-end Atmel MCUs. We discovered early on that although this MCU might have the capacity to run a lightweight version of the stack we did not want to begin our work by manually simplifying the stack and thereby adding another source of errors.

Instead we used the high-end ATMega128L, which we had received from ACTE.

4.2.1 Running the stack on an AVR

First of all the stack had to be configured to be as lightweight as possible by using the supplied program MakeApp. MakeApp helped us remove all profiles that we didn't need, such as SCO links.

It was also configured for an 8-bit system (which the AVR is) and for communicating with an Ericsson module. This communication is supposed to be standardized but in fact there sometimes still exists differences between modules from different manufacturers.

We soon noticed that compiling and running the stack was a big problem. We had not anticipated the amount of memory it required.

The ATMega128L has 128 KB of EEPROM[26], 4 KB of FLASH[27]and 4 KB of RAM[28]. To accommodate the stack's need for memory we had to change the heap[29], cstack [30]and return address stack [31]sizes.

---

[26] Saves the program code

[27] Preserves data between power-down/power-up (I.e. a small harddrive)

[28] Random Access Memory

[29] Dynamic memory allocation

[30] Saves constants

[31] Defines return pointers for functions

After some testing we set the heap to 0x949 (2 377) bytes and the cstack to a size of 0xA0 (160) bytes. We also set return address stack to 6 levels to be on the safe side.

A number of optimization done by the compiler was also necessary to shrink the code.

4.2.2 Adapting the hardware layer

Since the stack is running on a MCU the MCU has to be connected physically to the Bluetooth module. We did this via RS-232 and one of the built-in UARTs found in the MCU we used.

Adapting the hardware layer seemed simple enough when we browsed through the manuals supplied with the stack and the stack generation tool. There were three functions we had to implement in the stack for making it run on the AVR:

- Sending a byte via HCI (in our case RS-232).
- Receive a byte via HCI
- Generate an interrupt tick every 100ms (for timers etc)

We began by creating a small project without the stack where we could send and receive byte to a simple terminal program running on our host computer. We also created the timer to send a byte every 100ms to make sure it worked as promised. This worked satisfactory and we implemented it the same functionality in the full stack.

After a mail conversation with Ericsson we found out that the development board we were using was configured to accept data at a rate of 57600 bps, 8 data bits, 1 stop bit and no parity. We also had buffer overflow errors where the Bluetooth module would fill the buffers of the ATMega128. By implementing an interrupt routine which stored all received bytes in an array we could handle this error.

As soon as this was set in our program the board could communicate with the Bluetooth module.

4.2.3 Bluetooth hardware

This part is equally important for creating a cost-effective product as stack selection. We were not able to complete such a survey but it should be researched further if the X-Mouse is to enter mass production. There was no real incentive for us to study this issue since the market for Bluetooth modules changes swiftly and if the X-Mouse should reach mass-production this issue should be addressed at that point.

Despite this, we did some research of which companies could prove interesting in the future. If the X-Mouse would reach mass-production the bare minimum would be to contact the companies we identified. These are:

- Cambridge Silicon Radio
- Silicon Wave
- Atmel
- Texas Instruments
- Ericsson Licensing Technology

4.3 Debugging

One of our mayor problems was how to debug the actual hardware. As stated earlier we could identify most of the program logic errors by running the stack on a Win32 system.

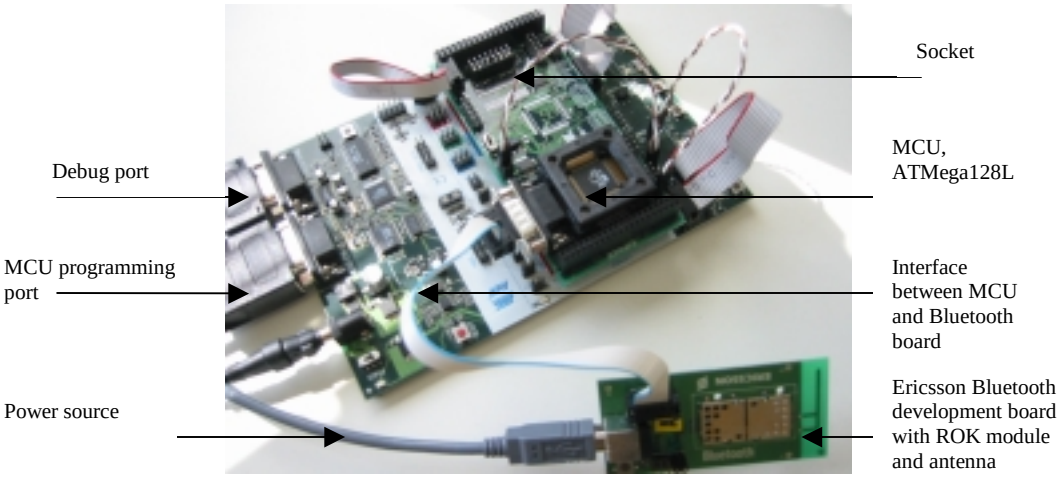


Figure 11: The development environment

The stack was well equipped with debugging information which was sent in runtime. Since our MCU was equipped with double UARTs we could send the debug information to one while communicating with our Bluetooth module on the other.

In theory this seemed to be an appropriate solution but in practice it was not possible to keep all debug information in the code and still fit it on the ATMega128L. After a short discussion we concluded that the most critical information we needed was stored in HCI-layer since most of our platform specific problems would probably occur while communicating with the Bluetooth module. We decided to strip down the debug information only to include the HCI-layer and only critical errors. This worked and we were able to receive data using a terminal program on our PC.

5 Discussion

5.1 Operating system drivers

As already stated we were not able to mimic the actual boot-up sequence of a mouse using RFCOMM and was therefore unable to create a solution where the operating system can search for the X-Mouse as a regular mouse.

Since by some odd reason virtual COM-ports are not included when searching for new devices (at least in Microsoft Windows) a new device driver is probably the way to go.

The roadmap for completing this step should be as follows:

1. Ensure why no information is sent. If it can be solved, do it.
2. If 1) is impossible create a device driver for X-Mouse

5.2 Bluetooth as a technology

Fact: Bluetooth is an immature technology so far. There are a number of incompabilities between modules from different manufacturers. The 1.1 standard was set only a year ago and so far manufacturers have problems implementing it. When using a mature technology like TCP/IP[32] it is assumed that such a stack will be 100% compatible with all other devices running TCP/IP. This is not an assumption that should be made in the Bluetooth world.

It is my prediction that Bluetooth still has at least 3-4 years of researching and testing before it can be regarded to be such a mature technology as TCP/IP. When working with Bluetooth always assume that modules and stacks from different manufacturers will be incompatible. Unluckily enough this currently means that most mayor companies go for the tried-and-tested solutions which leave little room for competition from smaller independent companies.

The Bluetooth SIG is currently doing a good job to ensure compability but it will take time before products start appearing on the market that has full compability, mainly

[32] Transfer Control Protocol / Internet Protocol, the standard protocol for Internet communications

because of the fact that Bluetooth is a complex technology. Even for the big players it takes time to create Bluetooth products.

5.2.1 Bluetooth in the future

From what I have learned during the completion of this thesis, I am convinced that Bluetooth will be an important technology in the future. Ad-hoc connectivity without any cable hassles will definitely attract even the worst laggards, but it does require the technology to be much more easy to use than of today.

5.3 Embedded Bluetooth

Our system for embedded Bluetooth can be used by APE in any of their upcoming prototypes or even live in a real product like the X-Mouse. This report proves that Bluetooth is available and it does not require huge amounts of money to implement it in a product.

The next step would be to study which Bluetooth module should be used in a live product. After a short market overview I can conclude that this will probably be one from CSR although this is subject to change. It should also be noted that CSR has a very interesting solution for HIDs on their roadmap. This solution should possibly be used. If not, our current Bluetooth system can be used as soon as IAR include the HID profile in their stack.

Ideally, this decision should be postponed as long as possible since the market for Bluetooth module is rapidly changing with many new big companies entering the market during the next few months.

6 References

6.1 Books

- Bray, J. et al**,** Bluetooth 1.1 – Connect without cables (Second Edition), Prentice-Hall Inc. 2002
- Muller, Nathan J, Bluetooth Demystified, McGraw Hill, 2001
- Kernighan, Brian W et al., The ANSI C Programming Language (Second Edition), Prentice-Hall Inc, 1988

6.2 Reports

- Dijkstra, Marcel & Martena, Albert R et al, *Radio Controlled Robot Car*, Master Thesis at University of Karlskorna/Ronney, July 2000
- Larson, Lars-Olof & Isaksson, Lennart, et al, *Remote measurements using Bluetooth wireless technology and client/server solution LOLLIS*, Master Thesis at Blekinge Institute of Technology, October 2000
- Gunée, Rickard & Iranpour, Ali, *PlayMobile*, Master Thesis at Royal Institute of Technology, November 2000
- Dahlberg, Anders & et al, *A Study of the Bluetooth Technology and Development of a Wireless Keyboard*, Bachelor Thesis at University of Blekinge, June 2000

6.3 Internet sites

- APE, http://www.apegroup.com
- ACTE, http://www.acte.se
- Cambridge Silicon Radio, http://www.csr.com
- Texas Instruments, http://www.ti.com
- IAR Systems AB, http://www.iar.com
- Blu2Space, http://www.blue2space.se
- Broadband Technology, http://www.broadband.se
- NoHau, http://www.nohau.se
- Codevision AVR, http://www.hpinfotech.ro/
- AVR Freaks community, http://www.avrfreaks.net/
- Developers free stack, http://www.cstack.com/
- IVT Corporation, http://www.ivtcorporation.com
- Widcomm, http://www.widcomm.com/

- Bluetoolsonline, http://www.bluetoolsonline.com
- Atmel, http://www.atmel.com
- Comtec Teleca, http://www.comtec.teleca.se/
- Information about opensource Bluetooth development http://opensource.nus.edu.sg/projects/bluetooth/links.html
- Information resource, http://www.infotooth.com/
- Information resource, http://www.thinkmobile.com/Wireless/Bluetooth/
- Imagecraft, makers of ICC AVR, http://www.imagecraft.com
- Atinav, http://www.atinav.com
- Mecel, http://www.mecel.se
- Lundinova, http://www.lundinova.se

6.4 Meetings

- Broadband Technology, Magnus Gustavsson, March 2002
- IAR Systems, Lars Sjöberg, April 2002
- NoHau, Johan Skyle et al., April 2002
- Lundinova, Marcus Persson & Anders Wahlström, April 2002
- Blue2Space, Linus Norrbom et al, April 2002

Appendix 1: Bluetooth profiles

The SIG currently has a total of 13 profiles defined. This section provides a short overview of their respective use.

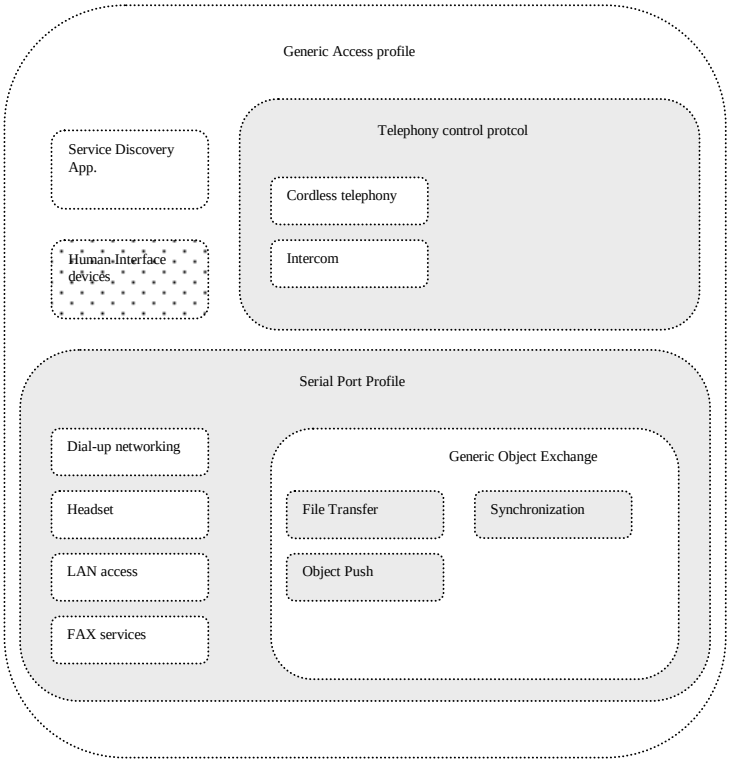


Figure 12: The Bluetooth profiles as of Bluetooth 1.1. Please note that the HID profile currently only is a working draft by the Bluetooth SIG.

6.5 Foundation profiles

Generic Access Profile: The most basic profile provides the base for all other profiles. Its purpose is to make sure all devices can establish a baseband link (see report for more detailed information).

Serial Port Profile: RS-232 cable emulation. For using legacy application over a Bluetooth link (see report for more detailed information).

Dial Up Networking (DUN): Allows a device (I.e. a PDA) to access a mobile telephone network or a PSTN via an access point (I.e. a mobile phone with modem).

FAX Profile: Similar to the DUN profile, but used to access fax services.

Headset Profile: Used for transferring voice calls between Bluetooth-enabled devices.

LAN Access Point Profile: Used for accessing a data network via an access point.
Generic Object Exchange Profiles (OBEX): Allows a device to push or retrieve data (objects) from another device.
Object Push Profile: Allows the pushing and retrieving of a limited number of objects from a device. Uses the OBEX profile.
File Transfer Profile (FTP): Like the Object Push but allows any form of data to be exchanged. Supports browsing of folders etc.
Synchronization Profile: Used for synchronizing the address book etc. of a PDA with that of a PC.
Intercom Profile: Point-to-point voice connections between Bluetooth headsets.
Cordless Telephony Profile: Allows a cordless phone to connect to a telephony base station.

6.6 Draft profiles

Human Interface Device (HID) profile: Allows devices such as a mouse, keyboard, joypad etc to communicate with a host (see report for more detailed information).
Hands-Free profile: Allows a mobile phone to connect to a hands-free device.
Basic Imaging profile: Used for cameras etc to transfer pictures with negotiated quality and size.
Basic Printing profile: Allows access to a printer and control of it without the need for a special driver.
Hard Copy Cable Replacement profile: Used when a device wants to connect to a printer without the use of the basic printing profile.

Appendix 2: Overview of ATMEL AVR microcontrollers

The AVR is an easy to use, low cost microcontroller with many important embedded features (I.e. UART, timer, analog comparator, etc). It features an 8-bit RISC processor running a single-cycle instruction set.

Atmel AVR MCU Product Selection Guide

	FLASH (KB)	EEPROM(Bytes)	RAM(Bytes) with 32 Register	Instructions	I/O Pins	Interrupts	Ext. Interrupts [1]	SPI	UART	TWI [4]	Hardware Multiplier	8-bit Timer	16-bit timer	PWM	Watchdog Timer	RTC Timer	Analog Comp	10-bit A/D Channels	On Chip RC-Oscillator	Brown Out Detector	In System Programming	Self Program Memory	Vcc(V)	Clock speed(MHz)	Packages	Available (unconfirmed)
ATtiny11L	1	-	32	90	6	4	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	2.7-5.5	0-2	8-Pin DIP, SOIC	now
ATtiny11	1	-	32	90	6	4	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	4.0-5.5	0-6	8-Pin DIP, SOIC	now
ATtiny12V	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	1.8-5.5	0-1	8-Pin DIP, SOIC	now
ATtiny12L	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	2.7-5.5	0-4	8-Pin DIP, SOIC	now
ATtiny12	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	4.0-5.5	0-8	8-Pin DIP, SOIC	now
ATtiny15L	1	64	32	90	6	8	1(+5)	-	-	-	-	2	-	1	Y	-	Y	4	Y[2]	Y	Y	-	2.7-5.5	1.6	8-Pin DIP, SOIC	now
ATtiny26L	2	128	128+32	118	16	11	1(+8)	1[6]	1[6]	1[6]	-	2	-	4	Y	-	Y	11	Y[2]	Y	Y	-	2.7-5.5	0-8	20-Pin DIP, SOIC 32-Pin MLF	Q3'02
ATtiny26	2	128	128+32	118	16	11	1(+8)	1[6]	1[6]	1[6]	-	2	-	4	Y	-	Y	11	Y[2]	Y	Y	-	4.5-5.5	0-16	20-Pin DIP, SOIC 32-Pin MLF	Q3'02
ATtiny28V	2	-	32	90	20	5	1(+8)	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	-	Y[3]	-	1.8-5.5	0-1	28-Pin DIP 32-Pin TQFP, MLF	now
ATtiny28L	2	-	32	90	20	5	1(+8)	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	-	Y[3]	-	2.7-5.5	0-4	28-Pin DIP 32-Pin TQFP, MLF	now
AT90S1200	1	64	32	89	15	3	1	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y	-	2.7-6.0	0-4	20-Pin DIP, SOIC, SSOP	now
AT90S1200	1	64	32	89	15	3	1	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y	-	4.0-6.0	0-12	20-Pin DIP, SOIC, SSOP	now
AT90S2313	2	128	128+32	120	15	10	2	-	1	-	-	1	1	1	Y	-	Y	-	-	-	Y	-	2.7-6.0	0-4	20-Pin DIP, SOIC	now
AT90S2313	2	128	128+32	120	15	10	2	-	1	-	-	1	1	1	Y	-	Y	-	-	-	Y	-	4.0-6.0	0-10	20-Pin DIP, SOIC	now
AT90LS2323	2	128	128+32	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	2.7-6.0	0-4	8-Pin DIP, SOIC	now
AT90S2323	2	128	128+32	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	4.0-6.0	0-10	8-Pin DIP, SOIC	now
AT90LS2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7-6.0	0-1	8-Pin DIP, SOIC	now
AT90LS2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7-6.0	0-4	8-Pin DIP, SOIC	now
AT90S2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	4.0-6.0	0-10	8-Pin DIP, SOIC	now
AT90LS4433	4	256	128+32	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	2.7-6.0	0-4	28-Pin DIP 32-Pin TQFP	now
AT90S4433	4	256	128+32	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	4.0-6.0	0-8	28-Pin DIP 32-Pin TQFP	now
AT90S8515	8	512	512+32	120	32	12	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	2.7-6.0	0-4	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90S8515	8	512	512+32	120	32	12	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	4.0-6.0	0-8	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90LS8536	8	512	512+32	120	32	16	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	2.7-6.0	0-4	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90S8535	8	512	512+32	120	32	16	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	4.0-6.0	0-8	40-Pin DIP 44-Pin PLCC, TQFP	now
ATmega8L	8	512	1K+32	130	23	18	2	1	1[3]	1	Y	2	1	3	Y	Y	Y	4+2 6+2	Y[2]	Y	Y	Y	2.7-5.5	0-8	28-Pin DIP 32-Pin MLF, TQFP	now
ATmega8	8	512	1K+32	130	23	18	2	1	1[3]	1	Y	2	1	3	Y	Y	Y	4+2 6+2	Y[2]	Y	Y	Y	4.5-5.5	0-16	28-Pin DIP 32-Pin MLF, TQFP	now
ATmega8535L	8	512	512+32	130	32	20	3	1	1[3]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-8	40-Pin DIP 44-Pin TQFP, MLF	Q3'02
ATmega8535	8	512	512+32	130	32	20	3	1	1[3]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5-5.5	0-16	40-Pin DIP 44-Pin TQFP, MLF	Q3'02

	FLASH (KB)	EEPROM(Bytes)	RAM(Bytes) with 32 Register	Instructions	I/O Pins	Interrupts	Ext. Interrupts [1]	SPI	UART	TWI [4]	Hardware Multiplier	8-bit Timer	16-bit Timer	PWM	Watchdog Timer	RTC Timer	Analog Comp	10-bit AD Channels	On Chip RC-Oscillator	Brown Out Detector	In System Programming	Self Program Memory	Vcc(V)	Clock speed(MHz)	Packages	Available (unconfirmed)
ATmega8515L	8	512	512+32	130	35	16	3	1	1[5]	1	Y	1	1	2	Y	-	Y	-	Y[2]	Y	Y	Y	2.7-5.5	0-8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega8515	8	512	512+32	130	35	16	3	1	1[5]	1	Y	1	1	2	Y	-	Y	-	Y[2]	Y	Y	Y	4.5-5.5	0-16	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega161L	16	512	1K+32	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	2.7-5.5	0-4	40-Pin DIP 44-Pin TQFP	now
ATmega161	16	512	1K+32	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	4.0-5.5	0-8	40-Pin DIP 44-Pin TQFP	now
ATmega162V	16	512	1K+32	130	35	20	3	1	2[5]	1	Y	2	1	4	Y	Y	Y	-	Y[2]	Y	Y	Y	1.8-3.6	0-1	40-Pin DIP 44-Pin TQFP, MLF	Q3'02
ATmega162L	16	512	1K+32	130	35	20	3	1	2[5]	1	Y	2	1	4	Y	Y	Y	-	Y[2]	Y	Y	Y	2.7-5.5	0-8	40-Pin DIP 44-Pin TQFP, MLF	Q3'02
ATmega162	16	512	1K+32	130	35	20	3	1	2[5]	1	Y	2	1	4	Y	Y	Y	-	Y[2]	Y	Y	Y	4.5-5.5	0-16	40-Pin DIP 44-Pin TQFP, MLF	Q3'02
ATmega163L	16	512	1K+32	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-4	40-Pin DIP 44-Pin TQFP	now
ATmega163	16	512	1K+32	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0-5.5	0-8	40-Pin DIP 44-Pin TQFP	now
ATmega16L	16	512	1K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega16	16	512	1K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5-5.5	0-16	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega323L	32	1K	2K+32	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-4	40-Pin DIP 44-Pin TQFP	now
ATmega323	32	1K	2K+32	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0-5.5	0-8	40-Pin DIP 44-Pin TQFP	now
ATmega32L	32	1K	2K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega32	32	1K	2K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5-5.5	0-16	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega64L	64	2K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-8	64-Pin TQFP, MLF	Q3'02
ATmega64	64	2K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5-5.5	0-16	64-Pin TQFP, MLF	Q3'02
ATmega169V[7]	16	512	1K+32	139	54	22	1(+16)	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	1.8-3.6	0-1	64-Pin TQFP, MLF	Q3'02
ATmega169L [7]	16	512	1K+32	139	54	22	1(+16)	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-3.6	0-4	64-Pin TQFP, MLF	Q3'02
ATmega103L	128	4K	4K+32	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	2.7-3.6	0-4	64-Pin TQFP	now
ATmega103	128	4K	4K+32	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	4.0-5.5	0-6	64-Pin TQFP	now
ATmega128L	128	4K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7-5.5	0-8	64-Pin TQFP, MLF	now
ATmega128	128	4K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5-5.5	0-16	64-Pin TQFP, MLF	now

[1] External interrupt + interrupt and wake-up on pin change
[2] High accuracy (5%) internal RC oscillator with programmable speed.
[3] Requires 12V signal on RESET pin during programming
[4] Compatible to I2C
[5] Programmable Serial USART
[6] Implemented by Universal Serial Interface (USI)
[7] On-chip LCD-driver included