CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Control Engineering

Bachelor Thesis

Bluetooth Stack for Embedded Systems

Supervisor: Ing. Michal Sojka
Author: David Plotek

Prague, 2009

Thesis assignment

Statement

The bachelor thesis was processed and written by myself. Only accessible and legal materials, which are referenced in list of references, were used. All source codes were implemented under General Public License.

In Prague 15th of January, 2009 ……………………………………

Author signature

Acknowledgement

I would like to thank to my parents, because they enabled me to study and supported me. Next thanks belong to my supervisor who supported me with new ideas and suggestions. He has been patient with me and explained many mechanisms and programming principles to me.

Annotation

The purpose of this paper is to implement Bluetooth stack for embedded systems. The Bluetooth represents a short-range wireless communication technology which is operating in unlicensed ISM bandwidth. This technology enables wireless connection for communication between the two devices – a computer as a control unit on the one side and a small robot (embedded system) on the other side. This paper analyses particular stack layers. Some of the layers were subsequently implemented in programming language C within the operational system Linux. The implementation of the HCI layers is described in detail and its functionality is demonstrated by the simple testing programm.

Anotace

Účelem této bakalářské práce bylo vytvořit softwarový Bluetooth stack pro embedded systémy. Bluetooth představuje bezdrátovou komunikační technologii na krátké vzdálenosti, která je provozovaná v bezlicenčním pásmu 2,4 Gz. Tato technologie umožňuje bezdrátové spojení pro komunikaci mezi dvěma zařízeními – počítačem jako řídicí jednotkou na jedné straně a malým robotem (embedded systémem) na straně druhé. V práci byly analyzovány jednotlivé vrstvy stacku. Některé z vrstev byly následně implementovány v programovacím jazyce C v rámci operačního systému Linux. Implementace HCI vrstvy je detailně rozepsána a její funkčnost je znázorněna jednoduchým testovacím programem.

Table of contents

1 Introduction

There are many communication technologies in the world today. Indians used some of the oldest ones, such as smoke signals. Nowadays we are using modern satellite communication, very high speed wired communication and terrestrial wireless transmissions. Each of these technologies can be described by a series of layers and its interfaces. The fireplace and smoke, for instance, represent physical layers. Indians represent the highest layers. The interface between Indians and smoke is made up from the cover and removal of a cloth producing an on or off signal. Modern technologies are much more complicated and many of them are realized by computers and sophisticated software solutions. A set of layers and their interfaces is called communication stack or protocol stack. It is a particular software implementation of computer networking protocol suite.

One of these modern technologies is Bluetooth. This advanced technology enables fast and reliable wireless communication between two or more devices. Features like low energy consumption, high integration level and low-cost transceiver microchips result in Bluetooth's usability in variety of applications. It is most commonly used in laptops, mobile phones, personal digital assistants, digital video cameras and even printers. From our perspective, one substantial application is communication between a control unit and a robot via Bluetooth technology. We can imagine a computer or a mobile phone as a control unit, while the robot is like an embedded system. This is one of many projects covered by the department of control engineering.

The main goal of my bachelor thesis is to implement a reliable and simple Bluetooth communication stack. This stack has to be sufficient for all operations needed by the embedded system. A control application running on an embedded system is able to use basic interface functions like connecting to a new device, sending/receiving data, maintaining connections and disconnecting. The stack manages basic operations between Bluetooth module hardware and application software. There exist commercial Bluetooth stacks, as well as open-source ones. All of commercial solutions are expensive and non-transparent. Free solutions implemented in Linux core are too extensive and they are not suitable for small embedded systems. This simple stack for embedded systems follows the Bluetooth specification standards. Stack was implemented with modular structure while considering low performance of the embedded system.

My thesis consists of the following sections. Section one contains basic theoretical information about wireless communication, Bluetooth technologies and communication stacks. The second section contains information regarding stack implementation. These parts involve implementation of particular protocol layers, with the final results presented in the last chapter.

2 Bluetooth

2.1 Wirelesses technologies

Bluetooth is a wireless communication technology and is classified to WPAN (Wireless Personal Area Network) group. In this section I mention general information about other wireless technologies. More exact information is possible to find in [11] or [3].

In 1878 David E. Hughes transmitted Morse code by an induction apparatus. It was the beginning of wireless data transmission. At present wireless technologies are used in many applications. Main applications are terrestrial radio and television broadcast, VHF radio, remote control, Global Position System and wireless networking. Wireless communication may operate via radio frequency, microwave and infra red or laser. Radio frequency is lower than microwave frequency and is used by radio broadcast and VHF radio. Infrared communication is suitable for very short distance and direct transmission like a remote control, laptops, mobile phones and PDAs. Laser technology is used in medium length data links. Laser links maximal transfer distance is 2km and maximal transfer speed is 10Mbit/s. In modern time the microwave band has an essential significance. Most of the high-speed data oriented wireless technologies work in the microwave band.

Wireless transmission has many advantages and disadvantages. The most essential disadvantages are disturbance, echo and limited frequency band. On the other hand thousands of kilometers of wires and cables are expensive and not very durable.

Over the last fifty years there has been a tendency to connect everything with everything. It started in the military sector when countries started connecting via the Internet. Cities continued this trend, connecting together. At present there is an effort to link all households in the world. The set up of several links is called a network. Networks are divided into several groups according to their extent. The largest one is the internet. It is the network of networks. The smaller ones are called the LANs (Local Area Network) and the MANs (Metropolitan Area Network). LAN and MAN are standardized by the Institute of Electrical and Electronics Engineers in 802.x standards. Two lowest ISO/OSI model layers are specified by these standards.

The standard number 802.11 specifies the Wireless LANs which operate in 5 GHz and 2.4 GHz public spectrum bands. In the 1997 the first 802.11 protocol was defined, but the 802.11b was the first widely accepted one, followed by 802.11g and 802.11n. The important standard in which Bluetooth is incorporated, presents the standard number 802.15. This one specifies Wireless Personal Area Networks just like the Bluetooth is. Wireless networks defined by these standards operate in ISM (Industrial, Scientific, Medical) band. The ISM is a free 2.4 GHz band, where broadcast energy output is lower then 100mW. The ISM band is exactly 83.5MHz wide in Czech Republic. It starts at 2.400GHz and ends at 2.483GHz. The wireless links use two transport spread spectrum technologies. The name ‘spread spectrum’ comes from the fact that the carrier signals occur over the full bandwidth of a device’s transmitting frequency. A carried power is spread into a wide spectrum and the spread spectrum signals are highly resistant to narrowband interference. The first of spread spectrum technologies is Direct Sequence Spread Spectrum, or DSSS. The second one is Frequency Hoping Spread Spectrum, or FHSS.

The DSSS signal spectrum is artificially extended by adding a pseudo-random sequence known to both transmitter and receiver. General transmission speed is 2Mbit/s and full width of one band is 22MHz. It is possible to operate three independent transmissions simultaneously. FHSS is a method of transmitting radio signals by rapidly switching a carrier among many frequency channels, using pseudo-random sequence too. The FHSS method is more convenient for lower speed transmissions and its robustness is higher then DSSS. Wireless technologies overview is shown in table 2.1 below.

Table 2-1 Overview of wireless technologies

IEEE standard number	Technology name	Speed / Operate frequency	Basic features
802.11a	WLAN	54Mbit/s / 5GHz	DSSS, indoor/outdoor use, max range 5km
802.11b	WLAN - WiFi	11Mbit/s / 2.4GHz	DSSS, indoor/outdoor use
802.11g	WLAN - WiFi	54Mbit/s / 2.4GHz	DSSS, enhanced speed
802.11n	WLAN	600Mbit/s / 5 or 2.4GHz	DSSS, may support existing b and g standards
802.15.1	WPAN – Bluetooth v.2.0	2.1Mbit/s / 2.4GHz	FHSS, indoor use only ,max range 100m
802.15.4	WPAN - ZigBee	250kbit/s / 2.4GHz	DSSS, indoor use only, max range ~50m

2.2 Bluetooth technology

Every person knowing technical innovation has heard about the Bluetooth already. The technical term like Bluetooth is well known in general public. Every adolescent uses this expression ordinarily in a tram or metro when he boasts about his new mobile phone in front of his schoolmates. The technology has been around for ten years at least. But many people who use this expression know nothing about or have very little knowledge of the technology and its applications. The purpose of this chapter is to explain, what Bluetooth actually is and how it works. It is also possible to find out more detailed information in references [1] and [3].

The simplest definition says that **Bluetooth technology** is a short-range wireless radio technology that allows electronic devices to connect to other devices. How long the distance can be, depends on the Bluetooth class and version. It was originally conceived as a wireless alternative to classical data cable serial links. Because a lot of cables on a table are too messy, the new technology thus provided the user with much more comfort.

The Bluetooth specifications are developed and licensed by **the Bluetooth Special Interest Group (SIG).** This organization was founded by the computing and telecommunications companies, such as Ericsson, IBM, Intel, Toshiba and Nokia. Microsoft, Motorola and others joined later. The organization is primarily run by a volunteer staff consisting of companies‘members. Every newly invented device using Bluetooth has to satisfy technical specifications formulated by the SIG and after that can be licensed.

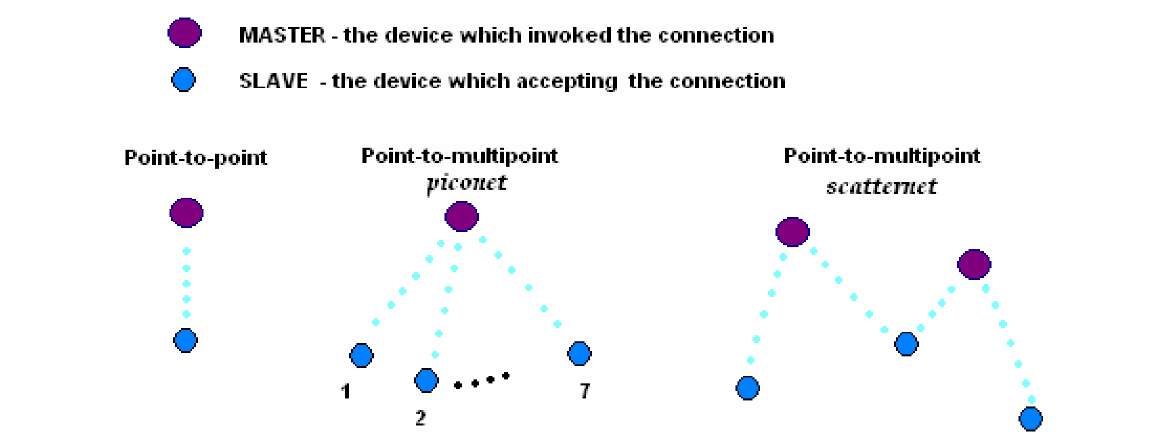
The **word ‘Bluetooth’** got its name after the 10th century Danish King Harald Bluetooth who united the previously warring tribes from Denmark and Norway. There is obvious similarity between history name and Bluetooth technology at present.

The **Bluetooth technology** was first developed in Scandinavia and unites diverse economic sectors such as computing, cell phones or automotive industry. The Bluetooth logo came from the Runic alphabetic and is composed from ‘H’ and ‘B’ characters.

As mentioned in the previous section, the Bluetooth devices use an unlicensed ISM bandwidth. This 83,5MHz wide bandwidth is totally free, but on the other hand, it is more disturbed by similar wireless technologies as WiFi. In addition, all microwave ovens are really big disturbers too. The Bluetooth technology is able to avoid interference by using the FHSS transmitting method. Thanks to the FHSS, a Bluetooth device can change the transfer channel very quickly and so is the chance to meet with another device on the same channel minimized. Based on f = 2042 + k [MHz], k = 0….,78 formula, there is 79 frequency channels and each one is 1MHz wide. The maximum hop rate is either 1600 hops/s in case of two devices being are connected together or 3200 hops/s when one device is in exploration state. The device hops to a new frequency after transmitting or receiving a packet. It depends on the packet type and device state how long the time slot is (the frequency remains unchanged for a specific time period). The lowest and most essential part of the whole system is the physical channel. The Bluetooth physical channel is characterized by the combination of the pseudo-random frequency hopping sequence and other aspects. Up to eight devices can share the same physical channel. The TDD (Time-Division Duplex) is used for physical channel multiplexing.

The Bluetooth technology enables to set up a point-to-point connection between two devices or point-to-multipoint connection among more devices. The connection invocatory device is ordinarily identified as **a master device**. The connection accepting device is ordinarily identified as **a slave device**. The physical channel can be shared among several devices, but maximum count of acting slave devices sharing the same channel is seven. Formation of more than two devices sharing the same physical channel and keeping frequency synchronization is called **piconet**. One slave device can be shared among more piconets. This bigger formation is called **scatternet**. See the figure below.

Figure 2-1 Bluetooth connection structures


MASTER - the device which invoked the connection
SLAVE - the device which accepting the connection
Point-to-point
Point-to-multipoint
piconet
1
2
7
Point-to-multipoint
scatternet

The SIG defines three main device classes. Devices are classified according the low-cost transceiver microchip transmission power. Communication distance is strictly dependent on the device class. Power class 1 device is possible to manage its power control. The power control is used for a transmission power limitation and can be used for optimizing its power consumption. See the class overview table below.

Table 2-2 Device classes

Power class	Maximum output power	Nominal output power	Minimum output power	Communication distance
1	100 mW (20 bBm)	N/A	1 mW (0 dBm)	Long range devices, up to 100m
2	2.5 mW (4 dBm)	1 mW (0 dBm)	0.25 mW (-6 dBm)	Middle range devices, 10m
3	1 mW (0 dBm)	N/A	N/A	Short range devices, 10cm

Each Bluetooth device has its own 48-bits long Bluetooth device address. Ordinarily it is marked as 'BD_ADDR'. The address is divided into three fields. The Low Address Part is 24 bits long and is assigned by the SIG. Upper Address Part and Non-significant Address Part are 24 bits long and form a company identification number (from which device is produced). In fact, the Bluetooth device address is a little bit similar to Ethernet MAC address but it is not the same.

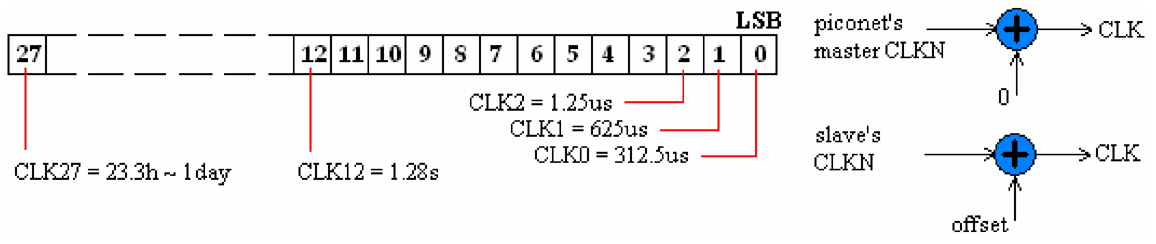
Figure 2-2 Format of BD_ADDR

LSB **MSB**

LAP						UAP		NAP			
0000	0000	0000	0010	0111	0010	1100	1001	1111	0110	1010	1100

Another essential feature of every Bluetooth device is its native clock which shall be derived from a free running system clock. The clock is generally implemented with a 28-bit counter. The least significant bit changes every 312.5µs giving a clock rate of 3.2 kHz. There are four important periods in the Bluetooth system, which are used for triggering of various system events. Each device has its native clock marked as 'CLNK'. The master clock 'CLK' is used for synchronization and shall be derived from master of the piconet.

Figure 2-3 Bluetooth clock counter, Derivation of CLK in master and slave

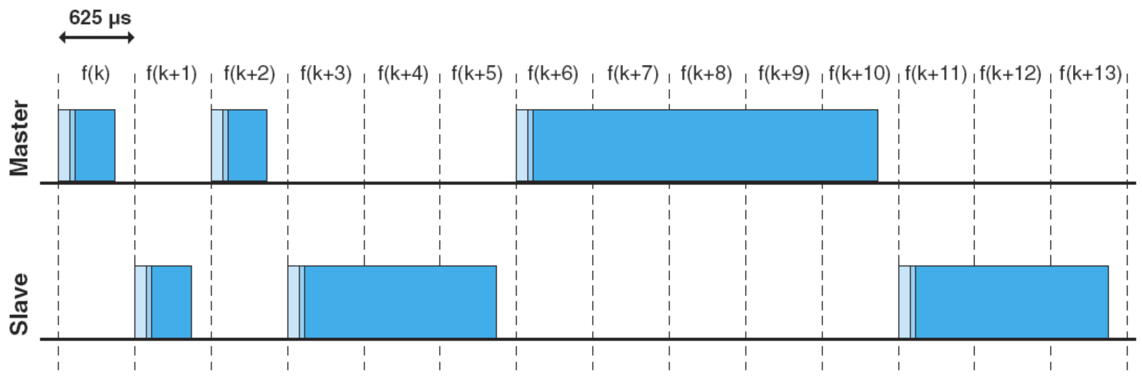


The physical channel is the lowest architectural layer in Bluetooth system. Every piconet uses its own physical channel and is capable to provide connection for up to seven devices excluding the master. Each physical channel is characterized by its own frequency hopping sequence which is determined by the native clock and a BD_ADDR part of the piconet master, by the specific transmission slot timing and by the access code and header which are used in every carried packet. When two devices want to transfer a packet with one another, it is necessary to be tuned on the same channel, to use the same frequency hoping sequence and the radio frequencies. It is also necessary for them to be within a nominal range. The four physical channels are defined:

- *Basic piconet physical channel* – used during the connection state, defined by master
- *Adapted piconet physical channel* – used for devices with adaptive freq. hopping
- *Page scan physical channel* – used for devices during connecting each other
- *Inquiry scan physical channel* – used for devices during searching each other

The basic piconet physical channel is divided into time slots. Every timeslot is 625μs long and has its own number ranging from 0 to 2^{27}- 1. In the Bluetooth systems, there are several packet types. Packets carrying longer data can occupy up to five single timeslots. In fact, one packet can occupy only the odd number of timeslots, because the TDD multiplexing forms a full duplex transmission via special mechanism where master sends packets in every even timeslot only and the slave is answering in every odd timeslot. In case of a five-slot packet, is the hop stopped for the period of five timeslots. Next frequency change (hop) is possible after time interval of 1.9ms. The figure 2-4 depicts the timeslots.

Figure 2-4 Multi-slot packets (one slot, three slots and fife slots)



In the Bluetooth system, it is possible to establish three basic logical transports for communication in both directions and two for broadcast purposes. The term physical link which is used in many references is not very suitable, but essentially it describes logical transport as well. The purpose of the Link Controller and the Link Manager is to establish and manage these physical links (For manager's description see page 17). These five transport types are defined as follows:

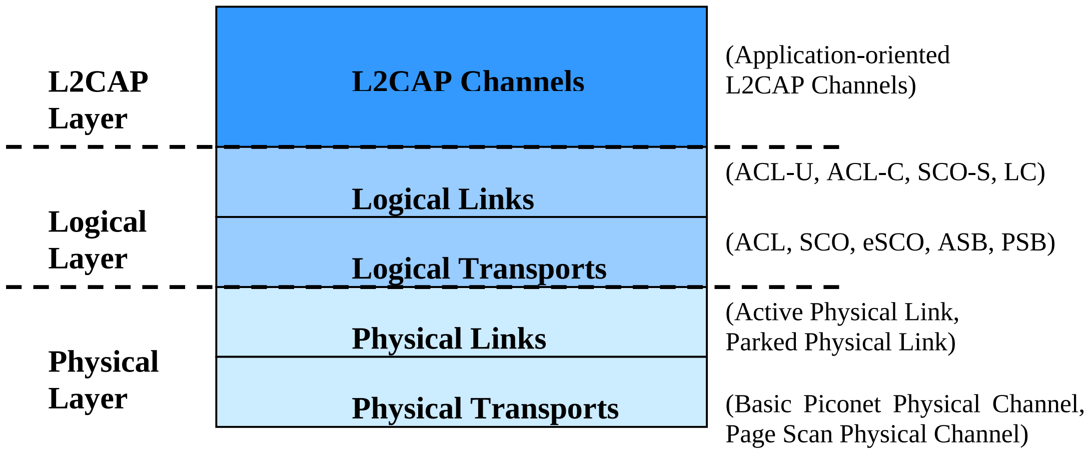
- *Synchronous Connection-Oriented logical transport* – SCO
- *Extended Synchronous Connection-Oriented logical transport* – eSCO
- *Asynchronous Connection-oriented Logical transport – ACL*
- *Active Slave Broadcast logical transport* – ASB
- *Parked Slave Broadcast logical transport* – PSB

The SCO logical transport is a point-to-point, synchronous and symmetric link between the slave and the master. "Synchronous" means that the time for sending and receiving a packet is strictly constant and the meaning of "symmetric" is that both-directional transport speed is identical. Standard symmetric transport speed is 64 kbps in each direction. The master may support up to three SCO links to the same slave. SCO links are used for time-dependent services like a voice and video, because the slot reservation mechanism offers registered real-time properties. It is considered as a circuit-switched connection between devices. Extended SCO supports asymmetric links and retransmission windows in addition.

The ACL logical transport do not reserve slots, but the master may establish ACL link with any slave on a per-slot basis. This link type supports symmetric connection with nominal both directional speed 433.9 kbps and an asymmetric connection with maximal speed 723 kbps in one direction and 57 kbps in opposite direction. For better data integrity, ACL link is able to ensure retransmission of corrupted data, but this lowers the transport speed. It provides a packet-switched connection between the master and all active slaves communicating in piconet. For better understanding, see the figure 2-5.

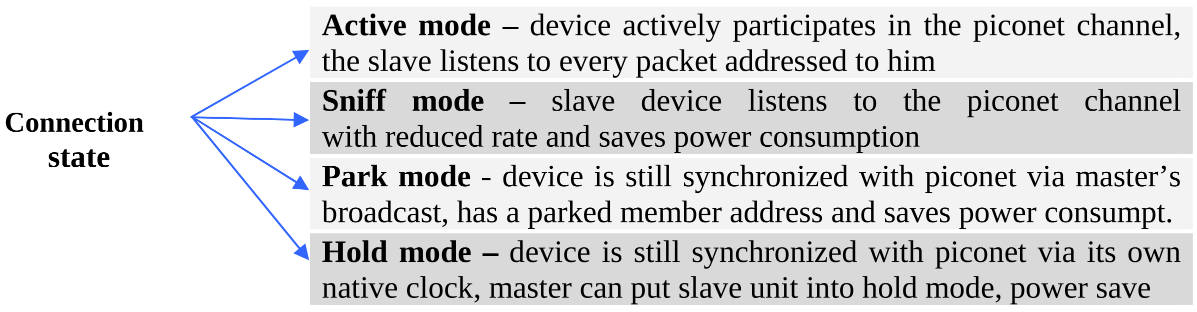
The last two types are used for master unidirectional communication with active and parked slaves. The ASB is used to transport L2CAP user traffic to all currently connected devices. The PSB is used for communication with parked slaves when master needs to resynchronize parked devices or when it needs to announce them to turn into active state.

Figure 2-5 Bluetooth generic data transport architecture


L2CAP Layer
L2CAP Channels
(Application-oriented L2CAP Channels)
Logical Layer
Logical Links
(ACL-U, ACL-C, SCO-S, LC)
Logical Transports
(ACL, SCO, eSCO, ASB, PSB)
Physical Layer
Physical Links
(Active Physical Link, Parked Physical Link)
Physical Transports
(Basic Piconet Physical Channel, Page Scan Physical Channel)

Each operative Bluetooth device can work in one of several states. Two main and the most usual are the Connection and Standby. The connection state has four state modes in which device keeps connection. There are additional seven substates in which the device obviously does not spend a long time period.

The **Standby** state is default for each device. Only the device native clock operates in this state. In **Connection** state, the device should be in one of the four connection modes.



Active mode – device actively participates in the piconet channel, the slave listens to every packet addressed to him

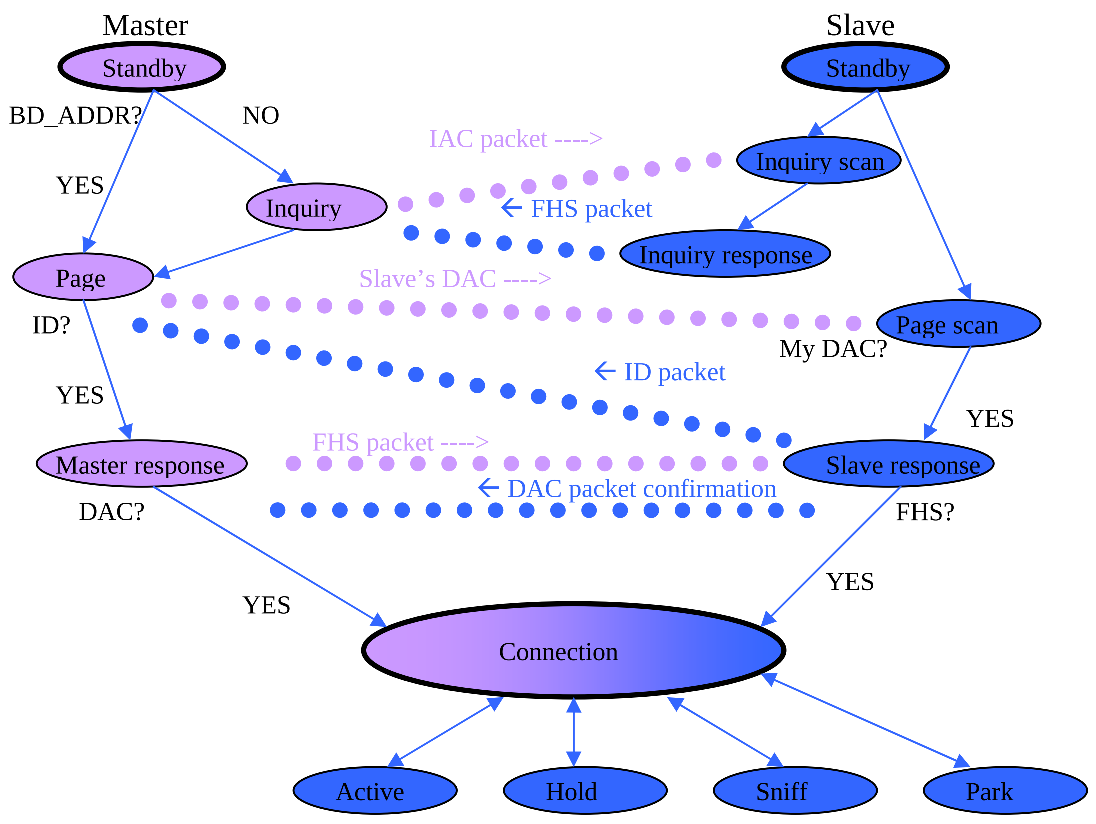
Sniff mode – slave device listens to the piconet channel with reduced rate and saves power consumption

Park mode - device is still synchronized with piconet via master's broadcast, has a parked member address and saves power consumpt.

Hold mode – device is still synchronized with piconet via its own native clock, master can put slave unit into hold mode, power save

In a Bluetooth system master and slave roles are not defined prior to a connection, the term ‘master’ is usually used for a device which as the first gets into page substate and the term ’slave’ is usually used for a device which as the first gets into page scan substate.

Figure 2-6 Scheme of main states and their substates during connection procedure



When a master device wants to make a connection to another slave device, there are two possibilities. The master device disposes of the slave’s address called ‘BD_ADDR’, for example from the past connection. It could directly get into page substate and try to set up the connection. The master device does not dispose of the slave’s address and has to search available slave devices in its range through inquiry substate. The slave device enables to be revealed or connected by listening to all frequencies and answering the master by suitable packet, such as FHS, ID or DAC. The whole connection procedure is well illustrated in the Figure 2-6. For more detailed information, see the Bluetooth specification [1] Controller volume, Part B, chapter 8 or well described states in [9] page 15 - 20.

2.3 Communication stacks

The purpose of this chapter is to explain what the communication stack actually is and how it works. It is not so easy to understand stack architecture. Technicians usually know about the ISO/OSI reference model for layered communications and computer network protocol design. But the Bluetooth communication stack is not too similar to this stack model reference. Exactly there are some related characteristics, but only for a few architectural blocks.

In many references and literature sources is written that the protocol stack or communication stack is a set of network protocol layers that work together. It also can be defined as a particular software implementation of a computer networking protocol suite. Let us analyze these technical terms. If one device wants to communicate with other device, it can be a complex problem to arrange all operations and make it operating well. It is most common thing to separate a complex problem into several partial problems in order to solve them more easily and separately. This complex problem, also called a protocol suite, is in communication terminology divided into several protocol layers. Each of these layers solves a set of problems involving the data transmission and provides services to the upper layer protocols which are based on using services from some lower layers. Upper layers are closer to the user application software and deal with more abstract data. Every upper layer uses the lower layer services and does not care about the data form. For example, two email clients do not care about the way how the data is transferred through the internet. Each of layers and suitable protocols is in charge of partial work. Each layer does the same work, but it can be done in many ways via different protocols.

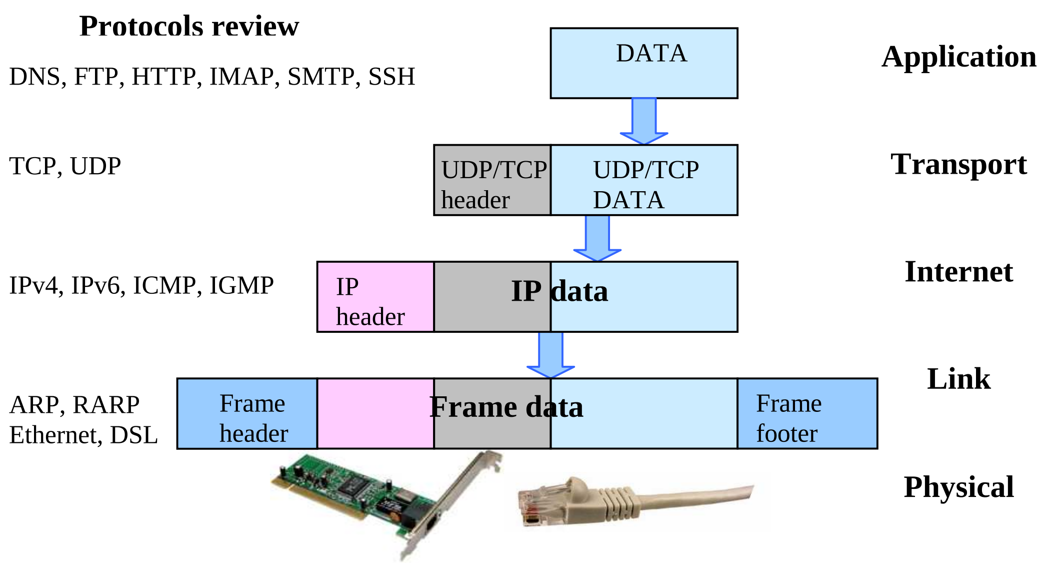
The International Organization for Standardization developed the Open System Interconnection Reference Model which is an abstract description for layered communications. It is the most common abstract protocol stack called the “ISO/OSI model”. It was developed to unify all communication architectures and to give a template how to implement communication stacks. The ISO/OSI model ordinarily consists of seven layers which are described in the table below.

Table 2-3 ISO/OSI abstract stack layers

Application	closest to the end user, interacts with software applications, determinates the identity and availability of communication partners
Presentation	formats and encrypts data to be sent across a network, establishes data context between application layer entities
Session	establishes, manages and terminates the connections between the local and remote applications
Transport	provides transparent transfer of data between end users, provides reliable data transfer services to the upper layer
Network	provides the functional and procedural means of transferring variable length data sequences from a source to a destination via networks
Data-Link	transfers data between network entities and detects and possibly corrects errors that may occur in the Physical layer
Physical.	defines the electrical and physical specifications for devices, defines the relationship between the device and the physical medium

Shortly speaking, this seven-layer model is the only the recommended concept for communication stack design. In fact, the protocol stack made up from strictly separate seven layers, has never existed. One of the most commonly used stacks in the world is the Internet Protocol Suite which is known as the TCP/IP. The name TCP/IP comes from the two important protocols which it contains: the Transmission Control Protocol and the Internet Protocol. These two were the first networking protocols defined in this suite. According to the way how TCP/IP model is implemented, the Internet Protocol Suite is generally divided into four layers. Each layer contains a number of various protocols providing different services. The TCP/IP suite uses encapsulation to provide abstraction of protocols and services. Every modular or layered suite uses this encapsulation method. Encapsulation is the characteristic feature of most networking models such as the ISO/OSI or TCP/IP model. It can be explained so that one layer protocol functions do not care about logical sense of layer above it and the protocols from layer below do not care about its functions and data. Universally the more abstract layer is often called the upper layer protocol while the more specific layer is called the lower layer protocol. See figure 2-7 where the encapsulation process is well explained.

Figure 2-7 Encapsulation sequence of user data in the TCP/IP protocol stack



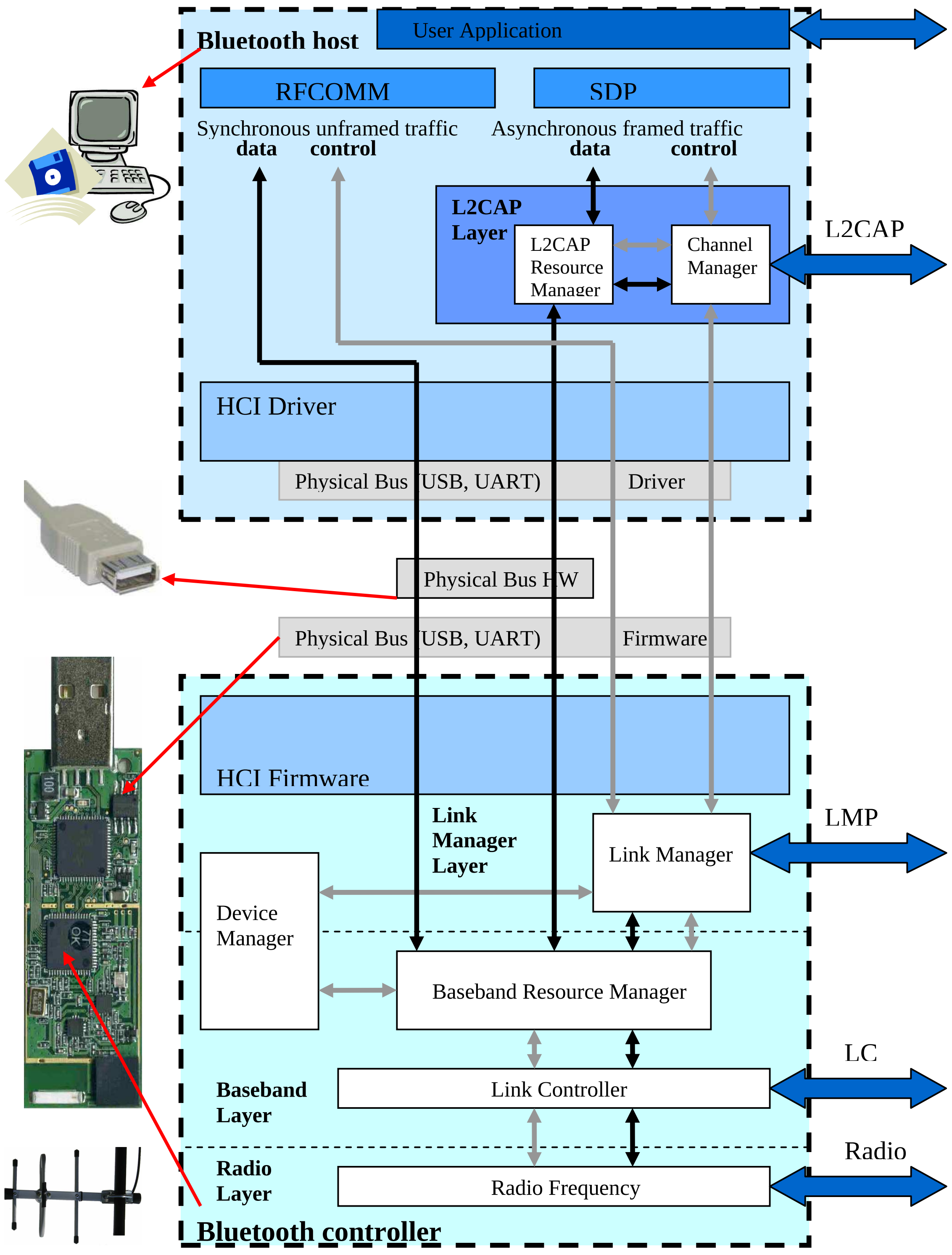
Apparently the TCP/IP does not respond to ISO/OSI model. These four layers are mapped to the seven-layer model by this way. The lowest Link layer usually matches to the OSI's Data link layer. The Internet layer is usually directly mapped to the OSI's Network layer. The Transport layer matches to OSI's same name layer. And finally the OSI's three top layers are merged and match to the TCP/IP Application layer. There is a lot of information about these stacks and their various protocols. Better explanation is out of range of this thesis. Very extensive information was written by famous author Jiří Peterka. His articles are referred in [10].

2.4 Bluetooth architectural layers

The right meaning of the term 'stack' is difficult to explain. It represents the software implementation of the specific protocol suit. The Bluetooth specification [1] gives only conceptional definition. All particular programming solutions are up to the user and his specific requirements. This chapter describes all Bluetooth layers as their conceptional definitions. The lowest core layers, sometimes grouped into a subsystem known as the Bluetooth controller are implemented by Bluetooth device manufacturers. The intermediate layer called the Host Controller Transport layer is situated between Bluetooth Controller subsystem and the Bluetooth Host subsystem. The Bluetooth Host subsystem consists of many layers: from the lowest called HCI Driver to the highest applications layers.

The figure below demonstrates the Bluetooth architecture scheme and its functional blocks relations.

Figure 2-8 Scheme of Bluetooth architecture



As a reader can see on previous figure, it is not very easy to understand the whole Bluetooth system. The system consists of many different protocols and functional mechanisms which are distributed in several layers. The Bluetooth Controller's parts are shortly described in next subchapter. More important Host Controller Interface characteristics and Logical Link and Adaptation Protocol features are described in more detail in the next sections.

2.4.1 Bluetooth controller's blocks

Device manager

This functional block is situated inside the Baseband Layer and the Link Manager Layer. The main purpose of this block is to control the general behavior of the Bluetooth device. For all operations, which are not directly related to data transportation, is responsible just the Device manager. For example connecting to other devices or inquiring for the presence of the other nearby devices. It also controls HCI commands and their effect on the system.

Link manager

The link manager is responsible for creation, modification and release of logical links. It communicates with the link manager in the remote device via Link Manager Protocol. This manager maintains logical link quality and stability by controlling of link, enabling of encryption on the logical transport or adapting of transmit power on the physical link.

Baseband resource manager

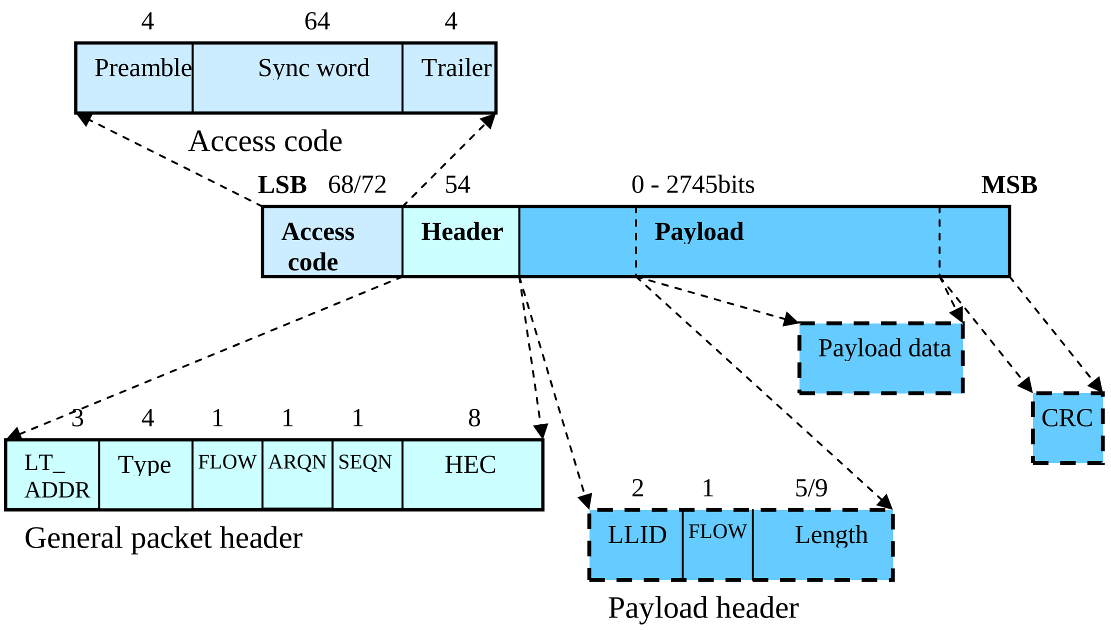
The baseband resource manager is responsible for the radio medium sharing. With the help of its scheduler it manages physical channel access contracts. It evaluates all surrounding functional blocks conditions and decides which request gets an access and which not.

Link controller

The Bluetooth packets are encoded and decoded by this controller. It takes care about data payload, physical channel parameters and logical transport parameters. It also creates the link control signaling, which is used to communicate flow control, acknowledgement and retransmission request signals.

The Bluetooth system uses the general packet format for data transmission. Every outbound and inbound packet transferred via physical channel is formatted into general packet format. The general packet bit ordering follows the little-endian format. Each packet consists of three entities which are the Access code, Header and Payload. All entities are composed from other sub entities. An Access code identifies all packets exchanged on the physical channel. All packets transferred via the same physical channel are preceded by the same access code. Packet header contains entities used by a link controller. Payload entity carries the upper layer control or user data. For better understanding see the figure below.

Figure 2-9 General packet and its entities format



The first entity access code is Preamble and it serves for DC compensation. The Synchronization word is derived from LAP (BT_ADDR main part) and is used for two devices synchronization. Its construction guarantees large Hamming distance between two synchronization words based on different LAPs. The Trailer is an optional entity and depends on access code type. There are two access code types. One is used during page scann and inquiry states (without Trailer) and the second is used during connection state (with Trailer).

General packet's header is only 18 bits wide, but from security reasons each its bit is used three times and it occupies 54 bits in total. LT_ADDR field presents the packet's logical transport address. Each piconet slave is distinguished by this address. Four bites long Type field is used for packet type signification. There are sixteen different packet types in Bluetooth architecture. Each packet type has its own purpose and usage. Common packets like ID, FHS, or DM1 occupy one time slot and are used for communication establishment. Other packet types are used for data transmission especially and usually occupy three or five time slots. Main packet types' overview is shown in the table 2-4. FLOW, ARQN and SEQN are transfer control state bits. HEC is the abbreviation for Header Error Check and is used for header integrity. Payload entities framed by dashed line are specific for each of logical transport types. The most important entity field used by ACL is LLID. LLID field helps to distinguish between start and continuation fragments used by upper L2CAP protocol.

Table 2-4 The most used packet types overview

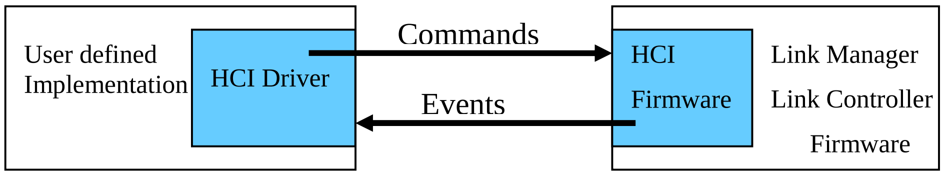
Name	Code	Slots	Usable Payload	Features
NULL	0000	1	0	Used to return link information when success or the RX buffer status, does not require confirmation
POOL	0001	1	0	Require confirmation from recipient, is used by the master in a piconet to poll the slaves
ID	na	1	0	Identification packet, consists of the device access code or inquiry access code, very robust
FHS	0010	1	0	Special control packet containing the BD_ADDR and sender's native clock
DM1	0011	1	1-18 B	Support control messages, 16-bit CRC,2/3FEC coding
DH1	0100	1	1-28 B	Similar to DM1, no FEC coding, 16-bit CRC
DM5	1110	5	2-226 B	Payload header 2B, 16-bit CRC, 2/3FEC coding
DH5	1111	5	2-341B	Similar to DM5, no FEC coding, 16-bit CRC
HV1	0101	1	10 B	No payload header, data bytes are protected with a 1/3FEC cod.
HV3	0111	1	30 B	No payload header, no FEC coding
DV	1000	1	80+150b	Combines data + voice, voice has no FEC, data has 2/3FEC cod.

🟪	Link control packets
🟩	ACL packets
🟦	Synchronous packets

2.4.2 Host Controller Interface

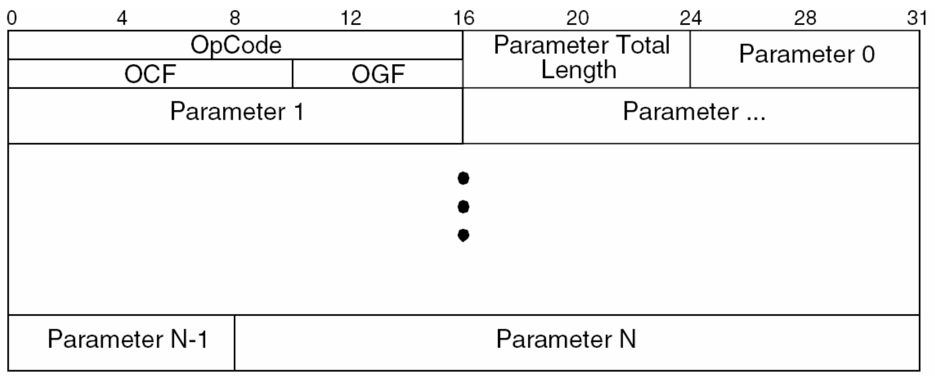
This interface is the lowest user accessible interface in Bluetooth layer architecture. It provides a uniform interface method of accessing the Bluetooth controller capabilities via defined commands. Through these commands user can create new logical connections, manage flow control or observe the Bluetooth controller. Bluetooth host and Bluetooth controller are connected via transport layer, as demonstrated in the figure 2-10. This transport layer is totally transparent for both surrounding layers. The Bluetooth host receives Bluetooth controller events as a feedback on its commands.

Figure 2-10 Commands and Events exchange



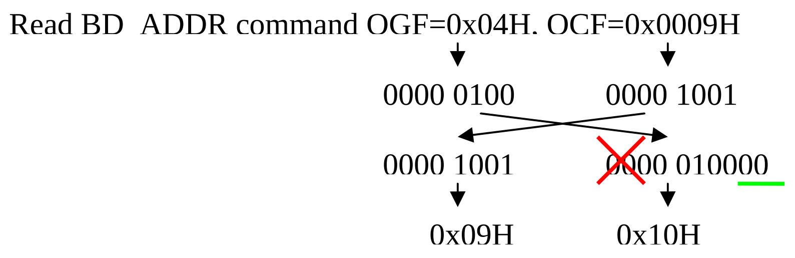
There are many events and commands used by HCI. Overview of all used events and commands in this thesis is enclosed in Appendices. All commands and events are listed in [1] Controller volume, Part E.

Figure 2-11 HCI command packet



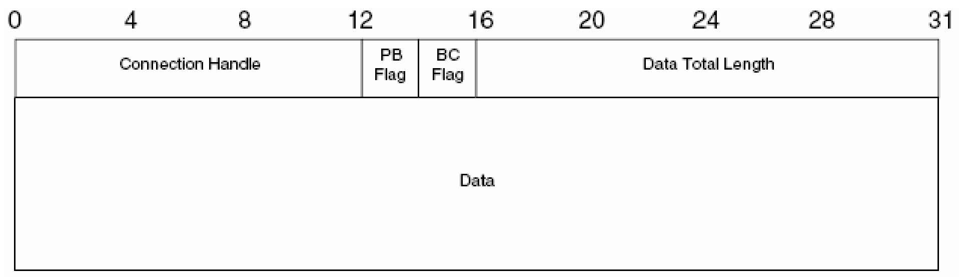
The figure 2-11 shows HCI command packet format. This packet length can be up to 255 bytes excluding command packet header. Each command has its own **OpCode** parameter for self-identification. The OpCode parameter is divided into two fields called the OpCode Group Field and OpCode Command Field. Additional fields represent command's parameters. In the Bluetooth system, there are six command groups and each contains several specific commands. The figure below shows how to create OpCode.

Figure 2-12 OpCode generation



The following figure shows the HCI ACL data packet format. These data packets are used to exchange data between the Bluetooth host and Bluetooth controller.

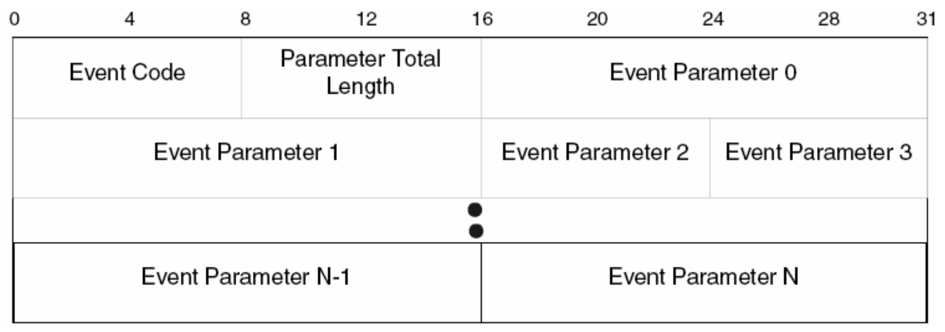
Figure 2-13 HCI ACL data packet



Each established connection between two remote devices has its own handler for self-identification. Because each device should have more then one established connection, it recognizes connections by handler. Connection handle field is used just for this purpose.

The HCI Event packet is used by the Controller to notify the Host when events occur. The Host must be able to receive this event which can be up to 255 bytes long. All Events start with Event code which serves for event identification. The parameter Total Length field presents parameters length in octets. Following fields are used for return command parameters.

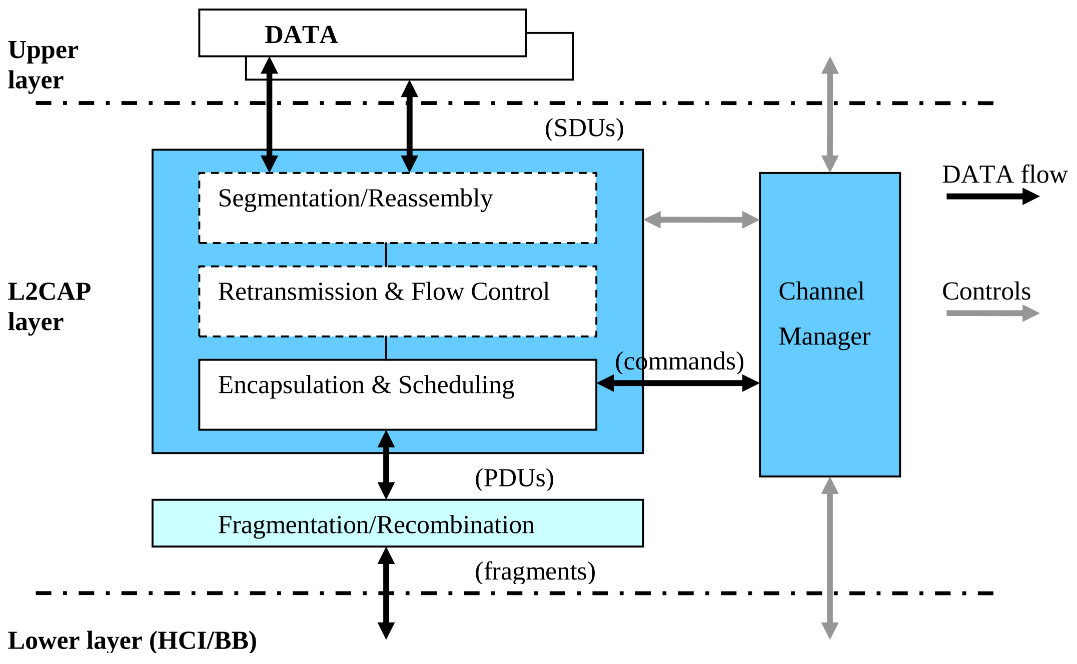
Figure 2-14 HCI Event packet



2.4.3 Logical Link Control and Adaptation Protocol

This protocol is situated above the baseband layer and provides connection-oriented and connectionless-oriented data services to upper layers protocols. The baseband physical ACL links are utilized by this protocol. The L2CAP is capable to accept data from different upper protocols like a SDP or RFCOMM via multiplexing capability. Other important features are segmentation and reassembly. It is possible to accept up to 64 kilobytes long data from upper protocols and split it into smaller segments which are long as a baseband packet size. This protocol provides quality of services by flow control, error control and retransmissions. Figure below shows the L2CAP functional blocks and their relationships.

Figure 2-15 L2CAP architectural blocks



3 Stack implementation

This section is focused on outcomes of my work and tries to provide sufficient description and explanation of my programming solutions. The first chapter gives a brief overview of existing stack solutions. The second chapter contains description of main software parts of which the stack consists. Several functional block diagrams show the main communication procedures. There are also included essential source code parts. The purpose of the next chapter is to show how to avail the practical results of my work to the user. The simple testing program is described here and the way how to compile it and execute it is here described too. Work summary and future planes are contained at the end of my thesis.

3.1 HCI Driver solution

My thesis assignment consists of several individual objectives which are included in the front part. The first objective was to acquaint with the Bluetooth technology and to present how individual system parts operate. It seems simple but it was not so simple to understand the whole Bluetooth system specification. Despite the fact I know a lot of information about Baseband, HCI and L2CAP layers, which are reported in previous chapters, I have to admit that I do not know all the features of the system.

The second objective of the thesis was to summarize all existing Bluetooth software stacks. Very short overview and discussion is mentioned in first chapter. There are many stacks, but only one or two are suitable for embedded systems. The third objective was partly fulfilled and is reported in following chapters. The HCI driver and its functions which are demonstrated by testing command line program are described there too. The testing program is able to be executed on personal computer with Linux operating system. The program is also able to communicate with attached USB Bluetooth device via Linux sockets. Through this device, it is possible to find another Bluetooth device and establish the connection with it. Functionality of the connection is demonstrated by simple text messages which can be sent from one device to another.

This HCI driver is ready for L2CAP protocol which has not been implemented yet. Despite the fact, I have spent a lot of time over developing this stack parts, I have to admit that all objectives have not been achieved yet. One of the solutions was wrongly designed because I did not understand the Bluetooth architecture well and my programming experience was not on sufficient level.

3.1.1 Other Bluetooth stacks overview

The users may wonder why it is necessary to implement a new Bluetooth stack software, when some operating stacks already exist. Majority of these stacks are daily used in our mobile phones or computers. They can be also used in health care institutes for patients' data collection. Some of them are actually suitable for embedded systems, so why do not use them for small mobile robots presenting our embedded systems?

There are several reasons why not. The **non open-source** commercial stacks are too **expensive** and **non-transparent** and there is no possibility to adjust some features for our purposes. Stacks implemented in assembler just for one microprocessor type, miss **feature-richness** and **flexibility.** Additionally, open-source stacks as a BlueZ are **too complex** and not suitable for embedded systems. The table 3-1 shows various stacks and their advantages and disadvantages.

Table 3-1 Bluetooth stacks overview

Name	Platform/ Operating System	Basic features
Microsoft WinXP stack	PC/Windows	Only for USB dongles, since Windows XPsp2, totally non-transparent and non-flexible
Widcomm	PC/Windows	First commercial Bluetooth stack for Windows
BlueSoleil	PC, Embedded/ Linux , Windows	Non open-source commercial product , widely used , it supports many profiles, useful GUI
Toshiba stack	PC/Windows	It was specially implemented for some Fujitsu Siemens, ASUS, Dell and Sony laptops, it support many profiles, non open-source
BlueZ	PC/Linux, Android	Most powerful and usable open-source stack, included in Linux kernel since 2.4.6 version, too complex and not suitable for embedded systems
Mezoev1.2	PC, Embedded / Windows, Linux	Is hardware-independent, modular solution, optimized API design, non open-source
iWRAP3	PC, Embedded / Linux	Commercial stack developed by BlueGiga company, support many profiles

BlueZ is the Linux open-source Bluetooth software stack and it was an inspiration for my thesis. It consists of many programs, functions and utilities which are written in C programming language. The BlueZ most essential source and header files are situated in the **bluez-libs-x.xx** Linux package. This package contains these files: the *hci.c* where the basic host controller interface functions are defined, *hci.h* where all command and events structures and all constants are defined, *bluetooth.c* and *Bluetooth.h* where the support functions and procedures are defined.

3.1.2 Main software parts

All my program source codes are written in C programming language and are included in appendix CD. This program version described in this section of my thesis is not the first and not the last I hope. My solution has two versions at least. First solution was incorrect and non-modular. My bachelor thesis supervisor helped me to propose better modular draft. All versions of my HCI Driver implementation are kept on GIT public repository on http://

The HCI Driver program consists of these files:

hciembeded.h: This header file contains all useful data type structures and constants. These structures form all commands and events data format. In fact, most part of this file content was copied from original file *hci.h*. It was not necessary to form new structures, because the content sense is given by Bluetooth specification.

bt_hw.c: The purpose of the functions in this file is to provide a direct access to physical bus. The physical bus is the Linux socket in this case. This file is supposed to be replaced by UART access functions in future. Functions Open, Close, Read and Write are implemented in this file.

bt_hw.h: This header file belongs to the source file with the same name and its function prototypes and constants are defined in.

tiny_bt_hci_cmd.c: Supporting, sending and commands forming functions are defined in this file. For example, supporting functions print or compares Bluetooth device addresses. Each command forming function forms the HCI command bit by bit. When the format of command is done, the command array pointer is passed to the sending function as a parameter.

tiny_bt_hci_cmd.h: This header file belongs to the source file with the same name and its function prototypes and constants are defined in. In addition hci_filter functions and important structures are defined here too.

tiny_bt_hci_core.c: Functional core of the HCI Driver is implemented in this source file. There are implemented hci_callbacks, requests forming and HCI Driver manager functions. Each of these types will be explained in following chapter.

tiny_bt_hci_core.h: This header file belongs to the source file with the same name and its function prototypes and constants are defined in. Several essential structures are defined in this file. Important structures which are used for return data and parameters storing are defined in this file too.

testapp.c: This source file contains a main function. Functions from this file forms a simple state machine which testing basic HCI driver commands. It is able to get the Bluetooth device name, find other devices in its nominal range, establish connection with slave device and send data.

The most essential data structures, functions, procedures and constants are displayed in next several paragraphs.

hciembeded.h:

```
/* HCI Packet types */
#define HCI_COMMAND_PKT            0x01
#define HCI_ACLDATA_PKT            0x02
#define HCI_SCODATA_PKT            0x03
#define HCI_EVENT_PKT              0x04
```

Each HCI packet format is marked by this identifier in front of its firs octet.

```
typedef struct{
        __u8 byte[6];
} __attribute__((packed)) bt_address;
```

The Bluetooth device address structure is formed by six bytes long field.

```
#define OCF_CREATE_CONN            0x0005
typedef struct {
        bt_address      bdaddr;
        uint16_t        pkt_type;
        uint8_t         pscan_rep_mode;
        uint8_t         pscan_mode;
        uint16_t        clock_offset;
        uint8_t         role_switch;
} __attribute__ ((packed)) create_conn_cp;
#define CREATE_CONN_CP_SIZE 13
```

An example of command parameters structure is displayed above. The first constant means the OpCode Command Field number. Total size in bytes is shown by second constant. There is a similar structure for each HCI event and command in *hciembeded.h* file.

bt_hw.c:

```
int hw_bt_read(__u8 *p_recbuf)
{
        int len;
        while ((len = read(hw_dd, p_recbuf,
                         sizeof(p_recbuf) * HCI_MAX_EVENT_SIZE)) < 0) {
                if (errno == EINTR)
                        continue;
                if (errno == EAGAIN)
                        return 0;
                perror("Device descriptor reading problem. \n");
                        return -1;
        }
        return len;
}
```

Read function is periodically called. The purpose of this function is to check the physical bus and fill up the buffer which is passed as a buffer pointer.

tiny_bt_hci_cmd.h:

```
#define INQUIRY_CMD_OP                  0x0104
#define INQUIRY_CANCEL_CMD_OP           0x0204
#define CREATE_CONNECTION_CMD_OP        0x0504
#define READ_BD_ADDR_CMD_OP             0x0910
```

An OpCode is expressed by these constants.

```
typedef struct{
        __u16 OCF_OGF;
        void *p_cmdp;
        __u16 cmdp_len;
} hci_cmd_request;
```

‘Hci_cmd_request’ structure is used as an input parameter for sending function. It contains OpCode, command parameters structure pointer and its size measured in bytes.

tiny_bt_hci_cmd.c:

```
int send_hci_read_bd_addr_cmd(void)
{
        hci_cmd_request creq, *p_creq = &creq;
        memset(p_creq, 0, sizeof(creq));
        p_creq->OCF_OGF = READ_BD_ADDR_CMD_OP;
        p_creq->cmdp_len = 0;
        if (send_cmd(p_creq) < 0) {
                perror("hci_read_bd_addr wasn't sent\n");
                return -1;
        }
        return 0;
}
```

One of the request forming functions is displayed above. This function does not has a return parameter. It calls the ‘send_cmd’ function displayed below.

```
 int send_cmd(hci_cmd_request *p_creq)
{
        __u8 array[p_creq->cmdp_len + 4];
        __u16 sw_opcode;
        int ii;
        sw_opcode = swap_2_bytes(p_creq->OCF_OGF);
        array[0]= HCI_COMMAND_PKT;
        memcpy(&array[1], &sw_opcode,2);
        array[3] = p_creq->cmdp_len;
        if (p_creq->cmdp_len > 0) {
                memcpy(&array[4], p_creq->p_cmdp, p_creq->cmdp_len);
        }
        for (ii = 0; ii < sizeof(array); ii++) {
                printf(" %x",array[ii]);
        }
        printf("\n");
        if (hw_bt_write(array, sizeof(array)) < 0) {
                perror("hw_bt_write problem\n");
                return -1;
        }
        return 0;
}
```

‘Send_cmd’ function forms command data into one-dimensional field and sends it to trough the Write function. The format of sending data is displayed on figure 3-1 below.

Figure 3-1 Read Bluetooth device address command format

	1 10	16	24
0001	0 0 0 0 1 0 0 1 0 0	0 1 0 0 0 0	0 0 0 0 0 0 0 0
Packet type	OCF	OGF	Parameter total length

tiny_bt_hci_core.h:

```
typedef struct{
        __u8 actual_status;
        __u16 id;
        __u8 evt_type;
        __u16 req_opcode;
        void (*p_callback)(void *p_arg, void *p_recbuf);
        void *p_data;
} __attribute__((packed)) expect_evt;
```

This structure is used for events management. The HCI driver registers the propriety event to the event's array when some command is sent. The program expects the answer from the controller. Every command is acknowledged by propriety events. The most usual events are Command Complete Event and Command Status Event. The most commands are confirmed by this event. This event data is consists of Event code, Status, Number of HCI Command Packets and Command Opcode field. Some error states can be reported through the Status field. The number of command packets which are allowed to be sent to the Controller from the Host. According the OpCode parameter the program should recognize the command which was confirmed by this inbound command status event.

```
typedef struct{
        __u16 con_id;
        __u8 con_state;
        struct hci_dev_info master;
        struct hci_dev_info slave;
        uint16_t handle;
        int socket_fd;
} connection_hci;
```

Connection_hci structure serves for storing the information about connection state. There are kept the master device address and the slave device address.

```
typedef struct{
        bt_address *p_address;
        void (*callback_app_read_bd_addr)(bt_address *p_address);
} read_bd_addr_data;
```

This structure is used for the storage of the application or L2CAP callback function pointer and needed data parameters which has to be delivered to the higher protocol.

tiny_bt_hci_core.c:

```
int tiny_bt_read_bd_addr(bt_address *p_dest, void(*callback_app_read_bd_addr)(bt_address
*p_address))
{
        expect_evt evt, *p_evt = &evt;
        req_state status = ESTABLISHED;
        read_bd_addr_data *p_data;

        memset(p_evt, 0, sizeof(evt));
        p_evt->actual_status = status;
        p_evt->id = evt_id++;
        p_evt->evt_type = EVT_CMD_COMPLETE;
        p_evt->req_opcode = READ_BD_ADDR_CMD_OP;
        p_evt->p_callback = &callback_hci_read_bd_addr;

        p_data = malloc(sizeof(read_bd_addr_data));
```

```
		p_data->p_address = p_dest;
		p_data->callback_app_read_bd_addr = callback_app_read_bd_addr;
		p_evt->p_data = (void *) p_data;
		add_evt_toarray(p_evt);

		if (send_hci_read_bd_addr_cmd() < 0) {
			perror("send_hci_cmd error\n");
			return -1;
		}
		return 0;
	}
```

All ‘tiny_bt_xx_xx_xx’ functions are called by higher protocol and serve for ‘expect_evt’ structures registration, needed data memory allocation and command request sending.

```
	static void callback_hci_read_bd_addr(void *p_arg, void *p_recbuf)
	{
		read_bd_addr_data *p_data = p_arg;
		read_bd_addr_rp *p_rp;

		p_rp = (void *) (p_recbuf + 3);
		memcpy(p_data->p_address, &p_rp->bdaddr, 6);
		p_data->callback_app_read_bd_addr(p_data->p_address);
		free(p_data);
	}
```

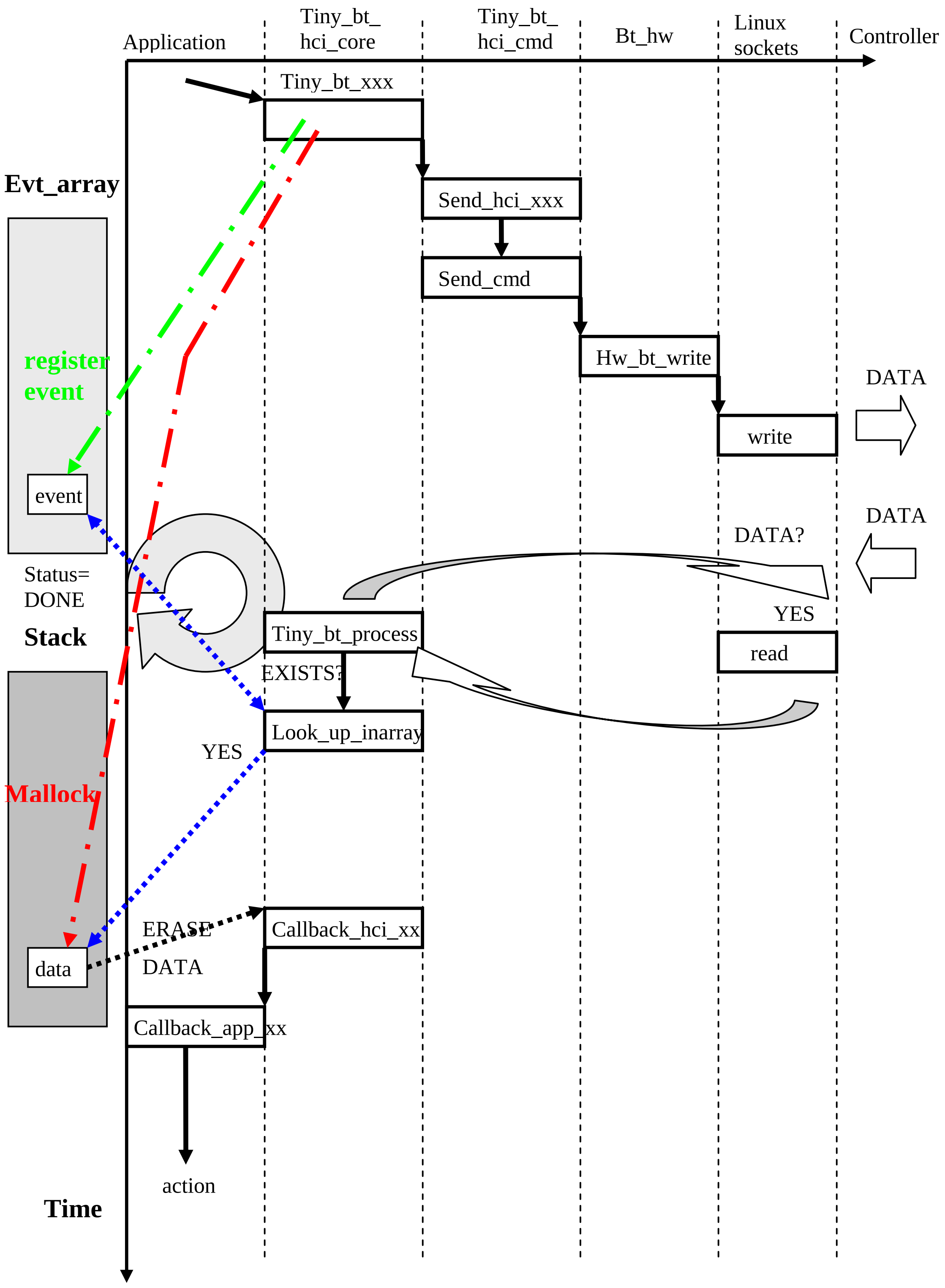
‘Callback_hci_xx_xx’ functions are called from ‘evt_array’ when the suitable event is accepted from the socket and is recognized in the array.

The tiny_bt_hci_core.c source file consists of other essential functions. These functions are too extensive and they will be explained shortly below. The first of them is the ‘tiny_bt_process’ function. This function is frequently called by the higher protocol in infinity while loop. Every time where the function is called, it calls the Read function and checks the socket. If the socket inbound buffer is empty, it returns back to the loop and tries to check it later. The function reads the data in the opposite situation and processed them through several conditions where the event type is compared with registered events. If the incoming event matches with some registered event, it calls the suitable ‘Callback_hci_’ function and this function calls upper protocol callback function. The registered event state is marked as DONE and its place in array can be replaced by another registered event.

Additional functions serve for event registering and event matching. These functions are called ‘add_evt_toarray’ and ‘look_up_inarray’. Functions are composed from common array operations.

3.1.3 Essential functional blocks

This subchapter tries to illustrate functions as a blocks and its data exchange.



3.2 Implemented stack using

4 Conclusion

The purpose of this paper was to implement a simple and reliable Bluetooth stack for communication between a control unit and embedded system. The control unit can be represented by a personal computer or a mobile phone, the embedded system is represented by small robot. The main objective of the thesis was to implement a communication between two USB Bluetooth modules through the Linux HCI sockets. Another objective was to import this solution into a small robot and to try its functionality.

In this thesis I have implemented the lower layer of the communication stack which ensures communication between a computer and the Bluetooth device. However, not all implementation phases were successful. The first unsuccessful implementation of HCI layer was done by non-modular way. Parts of the HCI driver were not divided into sections what resulted non-functionality of the code. After revision of the architecture, a new, modular solution was proposed which consisted of particular functions and call-back functions.

In the new solution, a part of the old solution was used and thanks to this, one part of the project already functions. This means that with the help of these functions, the device with one Bluetooth module is able to establish communication with another device and it is able to provide data transfer. The functionality of this solution can be proved by the program.

A shortcoming of the project was that the problem was too complex and required many technical skills. This cause that the program could be led into its final stage. In spite of this, I am convinced I have achieved the objectives set in the thesis assignment. By the means of this work, I had the opportunity to participate one of the projects covered by the Department of Control Engineering. After accomplishment in my diploma thesis, the work may become a part of the modern project directed to the communication with mobile robots.

References

[1] **TEAM OF CONTRIBUTORS.** *Specification of Bluetooth System.* **Revision v1.2.** http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

[2] **TEAM OF CONTRIBUTORS.** *LINUX, Documentation Project.* **1.revision. Prague: Computer Press, 1998. ISBN 80-7226-114-2**

[3] **TEAM OF CONTRIBUTORS.** *The comprehensive guide to everything Bluetooth related.* **BlueTomorrow.com, 2008.** http://www.bluetomorrow.com

[4] **HEROUT, Pavel.** *C language textbook.* **4.revision. Ceske Budejovice: Kopp, 2007. ISBN 80-7232-220-6**

[5] **DOSTÁLEK L., KABELOVÁ A.** *Velký průvodce protokoly TCP/IP a systémem DNS*. **2.revision. Prague: Computer Press, 2000. ISBN 80-7226-323-4**

[6] **DOSTÁL, Radim.** *Sockets and C++.* **Builder.cz, 2002.** http://www.builder.cz/serial147.html

[7] **DOSTÁL, Radim.** *Sockets and C/C++.* **Root.cz, 2003.** http://www.root.cz/clanky/sokety-a-c-mnozina-soketu/

[8] **MUŽIČEK, Petr.** *Model Remote Control.* **Diploma thesis. Prague, 2005**

[9] **PERMAN, Pavel.** *Model Remote Control.* **Diploma thesis. Prague, 2005**

[10] **PETERKA, Jiří.** *Archiv článků a přednášek Jiřího Peterky.* **eArchiv.cz, 2008** http://www.earchiv.cz

[11] **TEAM OF CONTRIBUTORS.** *Many individual articles aimed on communication technologies topics on Wikipedia***.** http://en.wikipedia.org/wiki

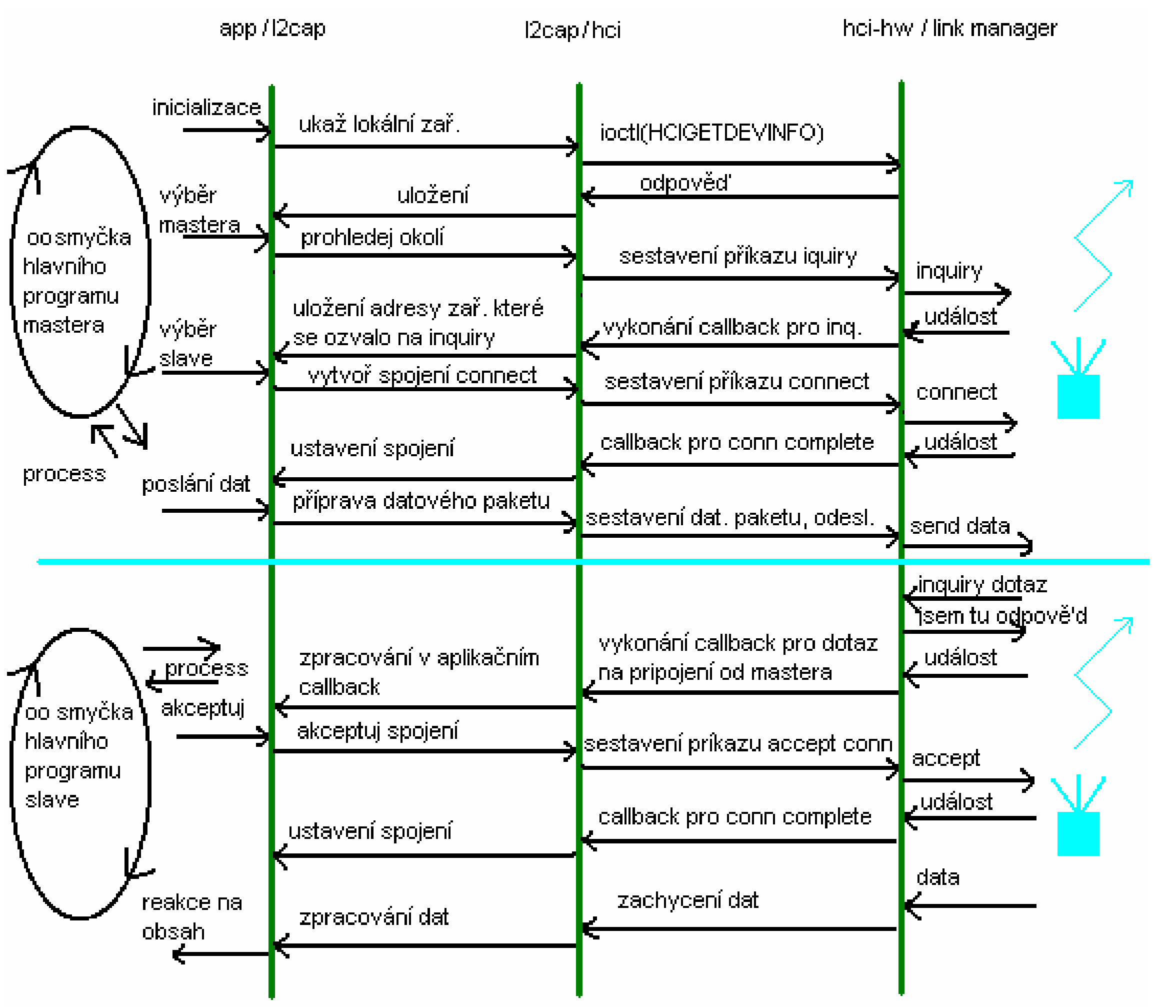
Shortcuts overview

GNU	General Public License
VHF	Very High Frequency
ISM	Industrial Science Medicine
PDA	Personal Digital Assistant
WPAN	Wireless Personal Area Network
LAN	Local Area Network
MAN	Metropolitan Area Network
WiFi	Wireless Fidelity
ISO/OSI	International Organization for Standardization / Open Systems Interconnection
SIG	Special Interest Group
IBM	International Business Machines
DSSS	Direct-Sequence Spread Spectrum
FHSS	Frequency Hopping Spread Spectrum
TDD	Time Division Duplex
MAC	Media Access Control
SCO	Synchronous Connection-Oriented logical transport
ACL	Asynchronous Connect-oriented Logical transport
ASB	Active Slave Broadcast logical transport
PSB	Parked Slave Broadcast logical transport
L2CAP	Logical Link Control and Adaptation Protocol
HCI	Host Controller Interface
LC	Link Controller
LM	Link Manager
TCP/IP	Transmission Control Protocol / Internet Protocol
ICMP	Internet Control Message Protocol
ARP	Address Resolution Protocol

RARP	Reverse Address Resolution Protocol
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
IMAP	Internet Message Access Protocol
SMTP	Simple Mail Transfer Protocol
UDP	User Datagram Protocol
SDP	Service Discovery Protocol
LMP	Link Manager Protocol
RFCOMM	Radio Frequency Communication
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver / Transmitter
LSB	Less Significant Bit
MSB	Most Significant Bit
LAP	Lower Address Part
UAP	Upper Address Part
NAP	Non-significant Address Part
HEC	Header Error Check
LLID	Logical Link Identifier
OCF	Opcode Command Field
OGF	Opcode Group Field

Appendices

Connection procedure between slave device and master device (in Czech)


app / l2cap
l2cap / hci
hci-hw / link manager
inicializace
ukaž lokální zař.
ioctl(HCIGETDEVINFO)
odpověď
oo smyčka hlavního programu mastera
výběr mastera
uložení
prohledej okolí
sestavení příkazu iquiry
inquiry
událost
uložení adresy zař. které se ozvalo na inquiry
vykonání callback pro inq.
výběr slave
vytvoř spojení connect
sestavení příkazu connect
connect
ustavení spojení
callback pro conn complete
událost
process
poslání dat
příprava datového paketu
sestavení dat. paketu, odesl.
send data
inquiry dotaz
jsem tu odpově'd
oo smyčka hlavního programu slave
process
akceptuj
zpracování v aplikačním callback
vykonání callback pro dotaz na připojení od mastera
událost
akceptuj spojení
sestavení příkazu accept conn
accept
událost
ustavení spojení
callback pro conn complete
data
reakce na obsah
zpracování dat
zachycení dat