



**Technical University of Crete**  
**Electronics & Computer Engineering**  
**Department**  
**Microprocessor & Hardware Laboratory**



Diploma Thesis

**Wireless Communication & Applications of  
BlueTooth, based on Microcontrollers**

**Politarhos Elias**

Supervising Professor: Professor A. Dollas

Committee: Professor A. Dollas  
Professor K. Kalaitzakis  
Assoc. Professor D. Pnevmatikatos

**June 2004**

**Chania**

## Microprocessor & Hardware Laboratory

"Be careful what you pretend to be because you are what you pretend to be."

Kurt Vonnegut Jr - Mother Night

Dedicated to my family

# Contents

---

Contents .....	2
1. Introduction .....	5
1.1. The Bluetooth® System .....	5
1.2. Thesis Stimulus, Scope & Results .....	7
1.3. Thesis organization .....	8
2. Relative Research .....	10
2.1. Bluetooth in our life.....	10
2.2. Bluetooth Products .....	12
2.2.1. Devices that support multiple connections.....	12
2.2.2. Simple devices for serial communication.....	13
2.3. Other wireless technologies .....	14
2.3.1. RF wireless communication .....	14
2.3.2. IR wireless communication .....	17
3. Existent & New Architecture .....	19
3.1. The Bluetooth protocol .....	19
3.1.1. Bluetooth Radio.....	21
3.1.2. Baseband .....	22
3.1.2.1. Bluetooth topology .....	22
3.1.2.2. Master-slave roles.....	23
3.1.2.3. Time Division Duplex in Bluetooth.....	24
3.1.2.4. Bluetooth connections .....	24
3.1.2.5. Packet types.....	25
3.1.3. Link Manager Protocol .....	26
3.1.3.1. Sniff mode .....	27
3.1.3.2. Hold mode.....	28
3.1.3.3. Park state .....	28

## Microprocessor & Hardware Laboratory

3.1.4.	Host Controller Interface .....	28
3.2.	The existing system (BlueApple-BlueBridge) .....	32
3.2.1.	Hardware .....	33
3.2.2.	Software .....	34
3.2.2.1.	BlueApple software architecture.....	34
3.2.2.2.	BlueBridge software architecture.....	38
3.3.	The new system: BluMiU architecture overview .....	41
4.	System organization .....	44
4.1.	BluMiU data transfer protocol.....	45
4.1.1.	The BlueTooth protocol approach.....	45
4.1.2.	The bluMiU protocol approach.....	46
4.2.	BluMiU hardware .....	48
4.2.1.	HOST.....	48
4.2.2.	BT module .....	50
4.2.3.	HCI & COM .....	51
4.2.4.	INPUT.....	51
4.2.5.	LEDs.....	52
4.2.6.	EXT DEV .....	52
4.2.7.	BluMiU hardware components' cost .....	52
4.3.	BluMiU software .....	53
4.3.1.	UART control modules .....	54
4.3.1.1.	UART reception decoders.....	55
4.3.1.2.	UART receiver (from the BT module) [U0RX] .....	57
4.3.1.3.	UART receiver (from the EXT DEV) [U1RX] .....	59
4.3.1.4.	UART transmitter control .....	61
4.3.2.	Command sending (to the BT module).....	63
4.3.3.	INPUT control software .....	65
5.	Testing & Validation.....	67
5.1.	BT module validation .....	67
5.2.	Software validation .....	69
5.2.1.	UART transmitter validation .....	70
5.2.2.	INPUT decoder validation .....	70
5.2.3.	UART decoder validation .....	71



## Microprocessor & Hardware Laboratory

5.3.	BluMiU validation.....	72
5.3.1.	BluMiU pre-connection setup.....	73
5.3.2.	Discovery of BlueTooth devices by the client .....	74
5.3.3.	Establishment of BlueTooth connections.....	75
5.3.4.	Data exchange .....	76
5.3.4.1.	The ability to exchange data.....	76
5.3.4.2.	Efficiency.....	78
5.3.4.3.	Possible improvements for higher data rates .....	79
6.	Applications.....	81
6.1.	File transfer.....	81
6.2.	FPGA programming.....	<b>Error! Bookmark not defined.</b>
7.	Conclusions and Future Work .....	85
7.1.	Conclusions .....	85
7.2.	Future work.....	86
	Appendix A.....	89
	ATmega161 AVR.....	89
	Appendix B.....	92
	The BT module .....	92
	Acronym List .....	94
	References.....	99
	Literature .....	99
	Internet Resources .....	99

# 1. Introduction

---

## 1.1. *The Bluetooth® System*

Wireless connectivity is a very desirable feature of nowadays' electronic devices. Radio was the first wireless technology and still is the fundamental element of virtually all modern wireless technologies. Lately, wireless communications between handheld, battery-operated devices and/or computers are thriving, since they contribute to reduce the amount of cables in the personal area. Though, the most important factor in wireless communications' popularity increase is probably the significant decrease in the cost of developing and obtaining devices that contain these technologies. Wireless personal communication adopts technologies that use the radio and the non visible light areas of the electromagnetic spectrum. One of the most recently developed personal networking technologies, which became quickly very popular, is Bluetooth technology. It initially began as a project to study the feasibility of a low-power and low-cost radio interface by Ericsson in 1994. [1]

In early 1998, the Bluetooth Special Interest Group (SIG) was formed by Ericsson, IBM, Intel, Nokia and Toshiba. [1] Now (June 2004) Bluetooth SIG is formed by 3com, Agere, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia and Toshiba, which are Bluetooth SIG promoter members, and a total of 3357 member companies [2] (promoters, associates<sup>1</sup> & adopters<sup>2</sup>). Bluetooth SIG developed Bluetooth wireless technology, which is a short range wireless communication system, intended to replace cables in the personal area. It has a maximum range of 100m and creates a small wireless network connecting portable and/or fixed Bluetooth enabled devices that exist in this range (Personal Area Network, PAN). Key features of Bluetooth are robustness, low cost, low power and small size. [3]

---

<sup>1</sup> Associate members have the opportunity to work with other Associate and Promoter companies on enhancements to the Bluetooth specification

<sup>2</sup> Adopter members just make use of Bluetooth technology

Today, 6 years later, Bluetooth system is one of the most popular systems for voice and/or data in every application where short range communication is needed (there is, however, a bandwidth limitation, 720Kbps). Widespread acceptance of the technology was helped by the truly open specification of Bluetooth (BTspec), which has been a fundamental objective of the Bluetooth SIG since its formation, and is promoted through the Bluetooth Specification Book (First stable version: 1.0b, latest: 1.2). That way every part of a Bluetooth enabled device should be qualified with the Bluetooth Specification; its hardware, the way it functions, its software and the way this device communicates with other Bluetooth enabled devices. When a device is BTspec qualified, it will be able to exchange data and/or voice worldwide with every other Bluetooth enabled device.

Concisely, BTspec defines a short (10m) or, optionally, medium (100m) range radio link capable for a data and/or voice communication. Its maximum data rate is defined to 720kbps<sup>1</sup> (kilobits per second), while every voice channel has a 64kbps data rate. The Bluetooth RF (physical layer) operates in the unlicensed ISM band, which is reserved for Industrial, Scientific & Medical purposes, at 2.4GHz. The system employs a transceiver with the following characteristics:

- Spread spectrum (2.402 to 2.48GHz)
- Full duplex transmission effect is provided through the use of a Time-Division Duplex (TDD) scheme
- Frequency hopping in a rate of 1600 hops per second

Connected Bluetooth devices in a Bluetooth network, which is called a *piconet*, use a specific frequency hopping pattern, which is algorithmically determined by characteristics of a Bluetooth device that acts as the Master of the piconet. The basic hopping pattern is a pseudo-random ordering of the 79 frequencies in the ISM band. The adaptive hopping technique improves Bluetooth co-existence with static (non-

---

<sup>1</sup> With the addition of the new EDR (Enhanced Data Rate) technology, Bluetooth transfer rates will reach a maximum of 2.1Mbps, three times the current rate. EDR is expected to become a mainstream standard by mid 2005

hopping) ISM systems when they are co-located. The output power of the BlueTooth device is 0dBm (1mW) for communication up to 10m (class 2 device) and +20dBm (100mW) for communication up to 100m (class 1 device). [4]

### **1.2. Thesis Stimulus, Scope & Results**

One of the most important and interesting capabilities of BlueTooth, which makes it stand out from other communication technologies in the PAN area, is that it enables multiple devices to connect simultaneously<sup>1</sup>. This fact, in conjunction with the kind donation by Ericsson Hellas to the Microprocessor & Hardware Laboratory (MHL) of the two BlueTooth modules that were used in this thesis and the development of an embedded applications platform of BlueTooth in the MHL, in the context of a previous diploma thesis by Christos Strydis [5], gave the inspiration for this thesis. In this thesis a second generation embedded application platform (bluMiU) has been developed. This platform provides data communication between a server and one or more clients and also a simple packet exchanging protocol suitable for very low processing power microcontrollers (microcontroller used: Atmel AVR ATmega161). The ultimate goal of this thesis is to wirelessly program an FPGA through this platform, but the specific design can be used wherever wireless transfer of data is needed and resources are truly limited.

Particularly, in the context of this thesis, the BlueTooth applications platform of the previous thesis (BlueAppleE) was redesigned from scratch so that it would provide an efficient and fully capable wireless data exchanging design.

The following modules were replaced:

- i) the data sending mechanism to the BlueTooth module (BT module),
- ii) the command sending mechanism to the BT module,
- iii) the decoding of the incoming event and data packets from the BT module mechanism and

---

<sup>1</sup> 7 active slaves & 255 parked slaves are supported by BTspec 1.2 and all versions of BTspec before it

- iv) the mechanism of data exchange between a device connected to the platform and the platform itself

The following elements were added:

- i) support for multiple device connections
- ii) interrupt based multiprogramming
- iii) a client-server architecture has been adopted
- iv) the host can be fully controlled from the device connected to the platform
- v) the device connected to the platform can give commands to the BT module
- vi) the platform reports the connected device of the addresses, the handles and the user friendly names of connected and inquired devices
- vii) the platform notifies the connected device whenever a disconnection occurs
- viii) a packet sending mechanism has been adopted for data transfers
- ix) connectionless data exchanging between connected clients and the server
- x) payload of data packets has been significantly increased. Now it is only limited by the buffer of the BT module or 65536bytes ( $2^{16}$ )

### **1.3. Thesis organization**

The rest of this thesis is structured as follows:

*Chapter 2:* The BlueTooth market, its inclinations and its future are presented in this chapter. Products that target in a more specialized audience are presented. Also, a comparison with “competing” technologies is made.

*Chapter 3:* Describes the lower levels of the BlueTooth protocol and the architecture of the existing design. It also reviews the existing design pros and cons and gives an overview of the new architecture.

*Chapter 4:* In this chapter the system designed is described in detail; the system’s architecture, its hardware and software are analyzed.

## **Microprocessor & Hardware Laboratory**

*Chapter 5:* The test procedures of the components of the system and the validation of the overall system are presented. A discussion commenting system efficiency and ways to improve the system ends the chapter.

*Chapter 6:* This chapter describes the applications that were developed to demonstrate the functionality of the system and propose future uses of it.

*Chapter 7:* In this final chapter the conclusions that can be extracted from this thesis are covered. The thesis ends with possible improvements on the system developed and proposes applications where it can be useful.

## 2. Relative Research

---

### 2.1. *BlueTooth in our life*

Although Bluetooth-enabled devices haven't quite entered the true mainstream yet, they are poised to take that next step. With Mobile phones, Personal Digital Assistants (PDAs), and headsets making significant strides over the last year, the automotive market beginning to make an impact, and Personal Mobile Gateway<sup>1</sup> products expected to emerge, shipments of Bluetooth-enabled manufactured equipment will experience a 60% CAGR (Compound Annual Growth Rate, The year over year growth rate of an investment over a specified period of time) between 2003 and 2008. [6] (Figure 2-1)

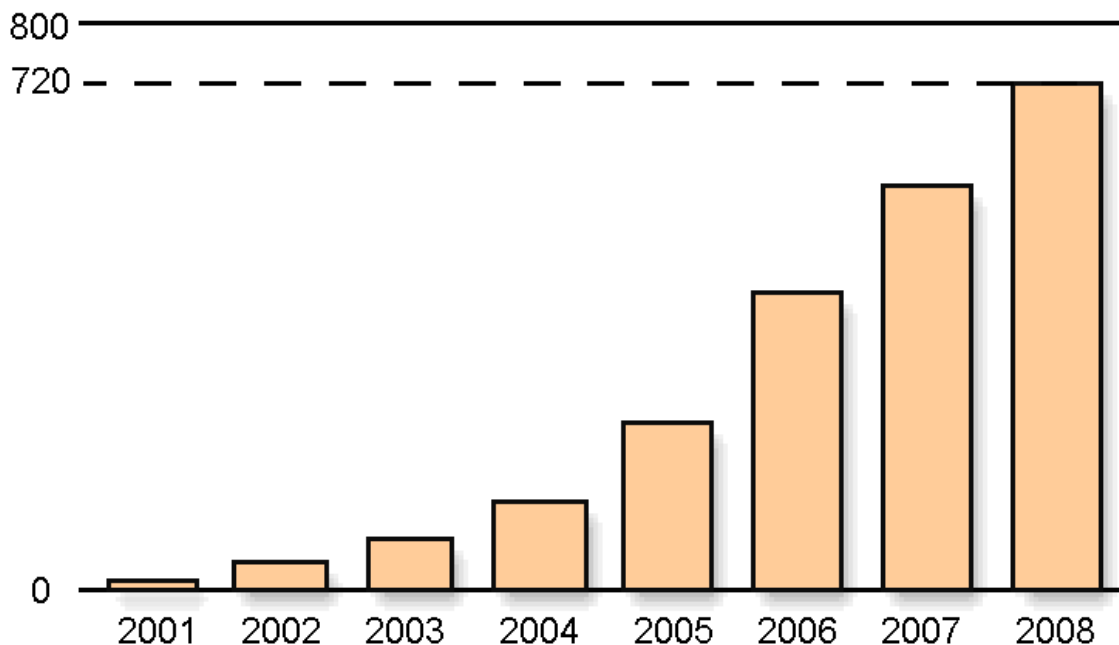


Figure 2-1 BlueTooth chipset forecast (Units in millions) (In-Stat/MDR, 4/04, [6])

---

<sup>1</sup> Personal Mobile Gateway: It is the point of connection between the wireless network and the new category of affordable and best-of-breed mobile devices (e.g. watches, pens, phones, messaging terminals, gaming devices, cameras) (<http://www.ixi.com/>)

While much of the focus of Bluetooth wireless communication is on the specification of the technology, the specification is actually rooted in a set of usage models. The specification itself was preceded by a marketing requirements document, which included usage scenarios that were an integral part of the objectives that the initial specification was to address. [1] These usage models are formally specified in the Bluetooth Profiles Book and help Bluetooth become a success in the mainstream market.

There is an unlimited number of Bluetooth enabled devices and Bluetooth applications, which intend to cover the communication needs of people around the globe. Every company targeting to the mainstream market and wanting to be in the cutting edge of technology, as far as communications are concerned, has gotten its hands dirty with Bluetooth technology, because they know that Bluetooth is very popular these days, thanks to the hard work of the Bluetooth SIG on marketing. It is essential for products targeting to a mainstream market to be easy to use and to have a user friendly interface. Therefore, these products tend to waste resources on various services that the Bluetooth protocol offers.

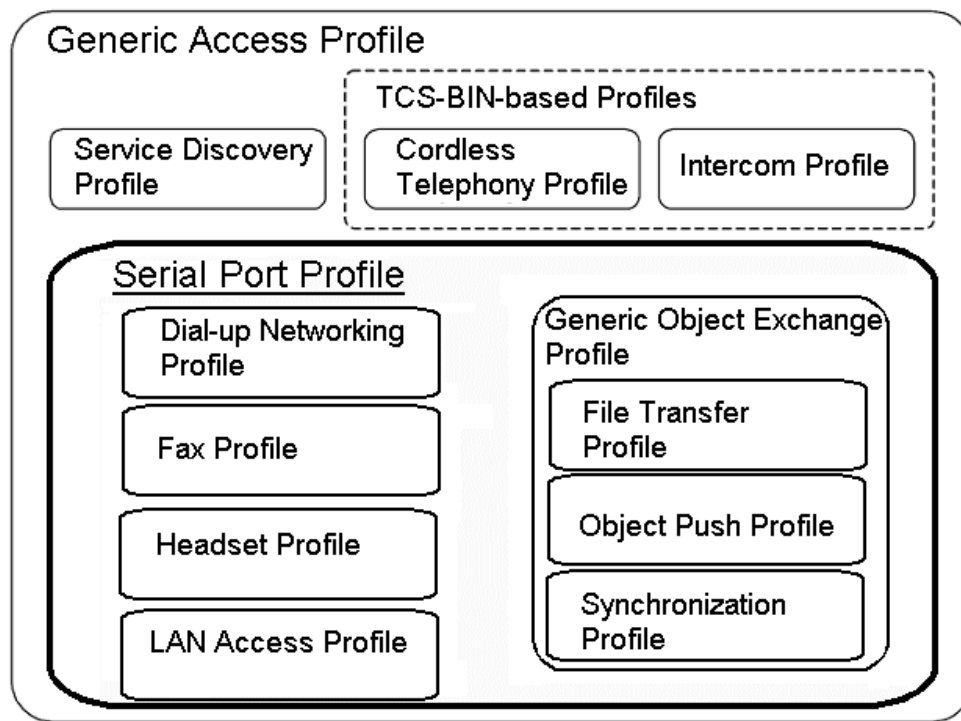


Figure 2-2 The importance of Serial Port Profile (BTspec 1.2 [3])



The most important and fundamental profile that the Bluetooth protocol offers is definitely the Serial Port Profile (SPP), that provides serial transfer of data between two Bluetooth enabled devices. This is the profile on which almost every other service provided by the Bluetooth protocol is based. [7] (Figure 2-2)

The main effort of this thesis is centered on the Bluetooth Serial Port Profile, because every other service can be developed in a layer above this thesis' platform, where limitations in resources might not exist.

## 2.2. *Bluetooth Products*

It is out of the scope of this thesis to present Bluetooth devices that exist in the mainstream market. They offer a huge amount of services, but -in general- waste resources, something that in many cases is not acceptable, especially when a large amount of services means more expensive equipment. There are, however, devices that provide valuable services, (even though their hardware resources are very low) which are targeted for the industrial market. Few of them even provide multiple serial port connections, which is exactly the purpose of this thesis. Some devices of stimulating interest will be presented next. They work under limited resources, in a hardware and software level, and provide serial connections to one or more devices.

### 2.2.1. **Devices that support multiple connections**

Only a few devices that provide multiple connections have been discovered, since most Bluetooth enabled devices are designed for the mainstream market, where multiple connections would just be something really complicated for the users. Some of the next devices can function as a network bridge between various kinds of networks. (LAN Access Profile) Their characteristics are presented below:

<b>Manufacturer</b>	<i>Bluenext</i>	<i>connectBlue</i>	<i>Stollmann</i>
<b>Device</b>	PROMI-MSP	SPA12i	BlueRS+E
<b>Connections</b>	1 to 7 or 1 to 35 <sup>1</sup>	1 to 3	1 to 3

---

<sup>1</sup> Depends on the model of the PROMI-MSP

## Microprocessor & Hardware Laboratory

<b>Data Rate</b>	723Kbps	300-921.6Kbps	2.4-230.4Kbps
<b>Profiles</b>	SPP, LAP, DUN	SPP, LAP, DUN	GAP, SDP, SPP
<b>Range</b>	10m~100m (w/ antenna)	10m	15m
<b>Others</b>	4USB ports, Networking support	-	Server mode, controllable by AT commands

### 2.2.2. Simple devices for serial communication

Some simple devices that provide serial connection to a BlueTooth enabled device without installing any additional software are presented next. They provide maximum security and are the optimal solution for serial cable replacement. Multiple-UART communication can be implemented in higher levels, as with normal serial cables.

<b>Manufacturer Device</b>	<b>Connection</b>	<b>Data Rate</b>	<b>Profiles</b>	<b>Range</b>
<i>AIRcable</i> Serial-to-Serial	2 predefined devices	4.8-115.2Kbps	SPP, LAP, DUN	10m
<i>Socket Com</i> Serial Adapter	1 BT device, Accepts AT commands	9.6-230Kbps	GAP, SDP, SPP	10m
<i>Wireless Futures</i> BlueWave	2 predefined devices	2.4-115.2Kbps	SPP	100m
<i>Brainboxes</i> RS232 BT	1 BT device, Accepts AT commands	2.4-115.2Kbps	SPP, DUN, FTP, OPP, FAX, LAN	100m
<i>TDK</i> blu2i RS232	1 BT device, Accepts AT commands	2.4-230Kbps	SPP, SDP	10m

The profiles mentioned are the following:

- GAP: Generic Access Profile
- SDP: Service Discovery Profile
- SPP: Serial Port Profile
- LAP: LAN Access Profile

- DUN: Dial-Up Networking profile
- FTP: File Transfer Profile
- OPP: Object Push Profile
- FAX: FAX profile

### **2.3. *Other wireless technologies***

BlueTooth was not the first attempt to develop a wireless technology for the personal area. There are some other technologies, which have the same scope with BlueTooth and are, until now, equally -and in some cases even more- popular. These technologies are communication through electromagnetic waves in the infrared light (IR) and radio frequency (RF) areas --RF is the technology that BlueTooth uses. A brief overview of these two technologies will be presented in the following sections, along with a comparison to BlueTooth, where applicable.

#### **2.3.1. RF wireless communication**

Technologies using Radio waves employ transceivers that can transmit and receive radio waves of a specific radio frequency. For communication in personal areas, low power transceivers are used so that they can only cover a distance of few meters; this means that they can cover as much space as it is needed for a PAN.

Due to the fact that the radio waves spectrum is limited and that most of the existing wireless technologies use them to function, governments around the world have legislated limitations and regulations to the use of the RF spectrum<sup>1</sup> and a license is needed for a technology to use a band of the RF spectrum. These limitations also define the power of the signal that this technology will use. All PAN networking technologies use bands of the RF spectrum that is agreed for them -in a worldwide basis- that their use will not require license, as long as these technologies' specifications cover some limitations, especially for the power of the signal they use. [1] For instance, in Europe and in the United States the 900MHz, 2.4GHz (ISM) and 5GHz

---

<sup>1</sup> NTIA Manual of Regulations & Procedures for Federal Radio Frequency Management:  
<http://www.ntia.doc.gov/osmhome/redbook/redbook.html>

bands do not require license for a technology to use them and, thus, it can be easily presumed that all RF technologies for PANs use one of these bands to function (the 5GHz band has been proposed for the Institute of Electrical & Electronics Engineers (IEEE) HIPERLAN standard [8]). The unlicensed bands mentioned above are:

- The 5GHz band is divided in band A, which covers the area between 5.15 and 5.35GHz, and band B, which is between 5.47 and 5.725GHz
- The 2.4GHz band: 2.4 - 2.48GHz (Until 2004, France and Spain were an exception in Europe, because they had narrower bands in the 2.4GHz band) and
- The 900MHz band: 902 - 928MHz

These bands are used by cordless telephones, microwave ovens, some remote controls and some wireless human interface devices for computers.

The 2.4GHz band is used by the really popular IEEE Wireless Local Area Network (WLAN) protocols: IEEE 802.11, 802.11b and 802.11g (a.k.a. WiFi). [9] Bluetooth also uses this RF band. Because the 2.4GHz RF band is really overcrowded by wireless protocols, some limitations for its use have been set, so that collisions between the signals of these protocols would be avoided and secure communication could be achieved. Bluetooth responded to these limitations by using a frequency hopping spread spectrum (FHSS) signal<sup>1</sup>. The rapid change in the transmit frequency reduces the chance of Bluetooth signals interfering with each other or with signals from other wireless networks. If a Bluetooth signal collides, the next time it will be transmitted it will be in a different frequency, so the probability to collide again is very small. Security is also enhanced by the frequency hopping because, if the Bluetooth transmission is intercepted, the next frequency that the Bluetooth system will use won't be known, thing that will probably confuse the interceptor.

Bluetooth and WiFi are two very well documented and specified protocols that have many similarities. Although, they have basic differences that will allow neither of them to replace the other, as happened with WiFi, which prevailed over HomeRF (a WLAN targeted home environments and users) in the beginnings of 2003:

---

<sup>1</sup> Bluetooth FHSS is explained in chapter 3.1.1 (page 21)

- BlueTooth has low power transmitters because it was designed for small battery operated devices like cell phones, PDAs, headsets and so on, that communicate in a radius of few meters. WiFi, on the other hand, was designed for computer networking in a maximum distance of 45m (802.11a) or 90m (802.11b and g). So, it does need stronger and bigger transmitters than BlueTooth to meet these requirements.
- WiFi is far more complex than BlueTooth. WiFi is designed to hook up an entire network, while BlueTooth is a cable replacement technology. This makes service discovery a simple task for BlueTooth, while WiFi requires the same degree of network management as any comparable wired network. It is this complexity of the protocol that makes WiFi unsuitable for use in devices like these that BlueTooth was designed for, that were mentioned earlier.
- BlueTooth is much slower than WiFi. BlueTooth has a maximum speed of 720Kbps, while WiFi can reach 100Mbps in 802.11g. This does not allow the use of BlueTooth as a LAN. [10]

BlueTooth (as IEEE 802.15.1™) was approved [11] in 15 April 2002 from Institute of Electrical & Electronic Engineers Standards Association (IEEE-SA) as a standard and it became an IEEE working group for wireless personal area networks (WPANs).

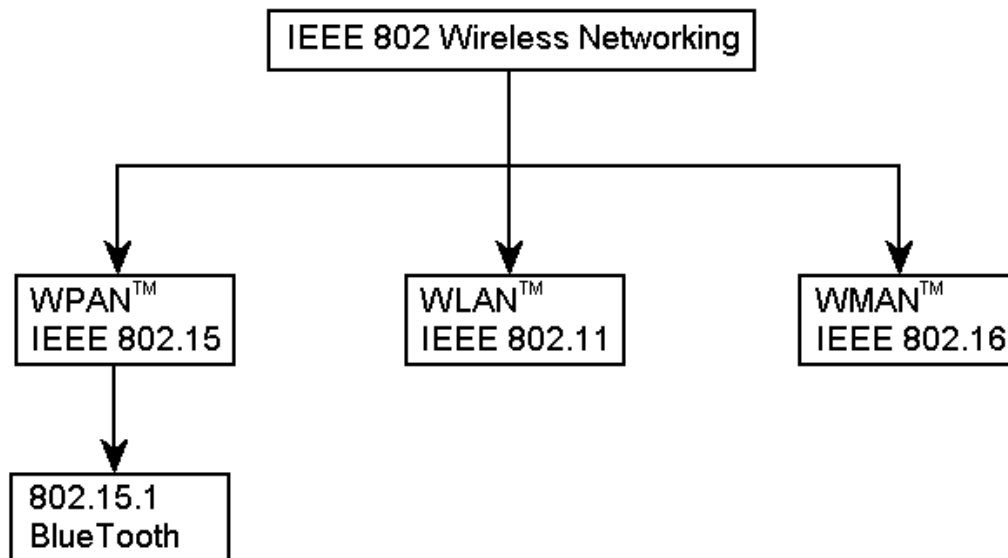


Figure 2-3 IEEE wireless networking standards

According to IEEE wireless networking standards, the following status quo has been established: 802.16 is the standard for Metropolitan Area networks (WMANs), 802.11

for Local Area Networks (WLANs) and 802.15 (BlueTooth) for Personal Area Networks. (Figure 2-3)

### 2.3.2. IR wireless communication

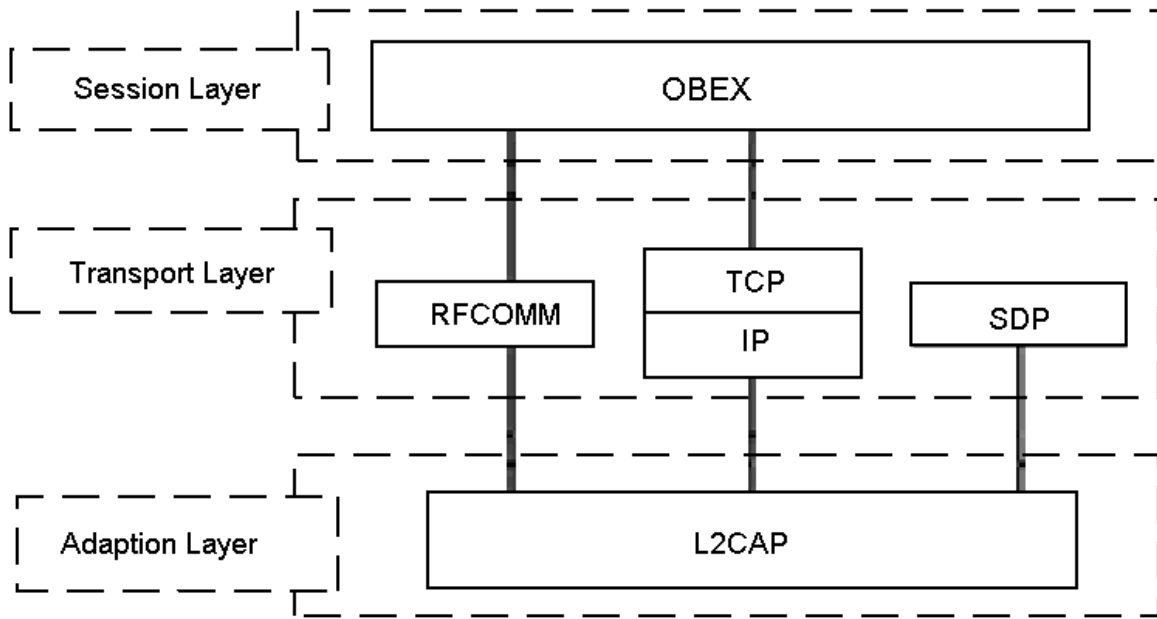
Short range wireless communication through infrared technology is probably the most common wireless technology for use in the personal space, since the vast majority of the remote controls used for home appliances, such as TVs, videos, air conditioners and HiFi systems -appliances that are somewhat essential in modern life-, use infrared technology. But infrared technology is also very popular in computers, computer peripherals, mobile phones and PDAs which use it for data exchange, in a way very similar to BlueTooth. Communication between these devices is specified by the Infrared Data Association (IrDA)<sup>1</sup>, which defines the hardware and software protocols for wireless communication intended to promote interoperable applications [1], just like the BlueTooth SIG.

BlueTooth and IrDA are very similar, since they were developed with the same scope, that is a short range, low power, low cost and unlicensed wireless communication, and are specified by very well documented standards that have a worldwide acceptance. Most manufacturers provide both of them with their products, allowing the customer to select the technology that fits his needs. The two technologies have their pros and cons, which help the customer to choose one of them:

- BlueTooth uses radio waves, while IrDA infrared light
- IrDA (16Mbps) is faster than BlueTooth (720Kbps)
- BlueTooth allows connections in a range of 100m in Class 1 BlueTooth devices, which is bigger than the range of IrDA (about 1m)
- IrDA is cheaper (1\$) than BlueTooth (estimated to reach 5\$)
- BlueTooth supports point to point and point to multipoint connections, while IrDA supports only point to point
- IrDA needs the two transceivers to be aligned and be in the other's line of sight, while BlueTooth can penetrate objects and doesn't need alignment [12]

---

<sup>1</sup> <http://www.irda.org>



**Figure 2-4 IrDA OBEX how it is reused in Bluetooth (BTspec 1.2 [3])**

IrDA and Bluetooth wireless applications share similar application domains, even though the underlying technology used to achieve usage scenarios is inherently different. Feature differences may cause one technology to be preferred over the other in certain environments and applications, although both have merit and both are likely to be deployed in pervasive computing devices. Thus the IrDA interoperability provisions of the Bluetooth specification can help to enable the best use of either or both technologies. The reuse of IrDA protocols<sup>1</sup> and specifically the infrared Object Exchange Protocol (OBEX) was identified as the design direction of the Bluetooth SIG early in the specification's development. [1] The purpose of the OBEX protocol is to enable the exchange of data objects and files. (Figure 2-4)

---

<sup>1</sup> IrDA had been already developed when Bluetooth research started

### 3. Existent & New Architecture

---

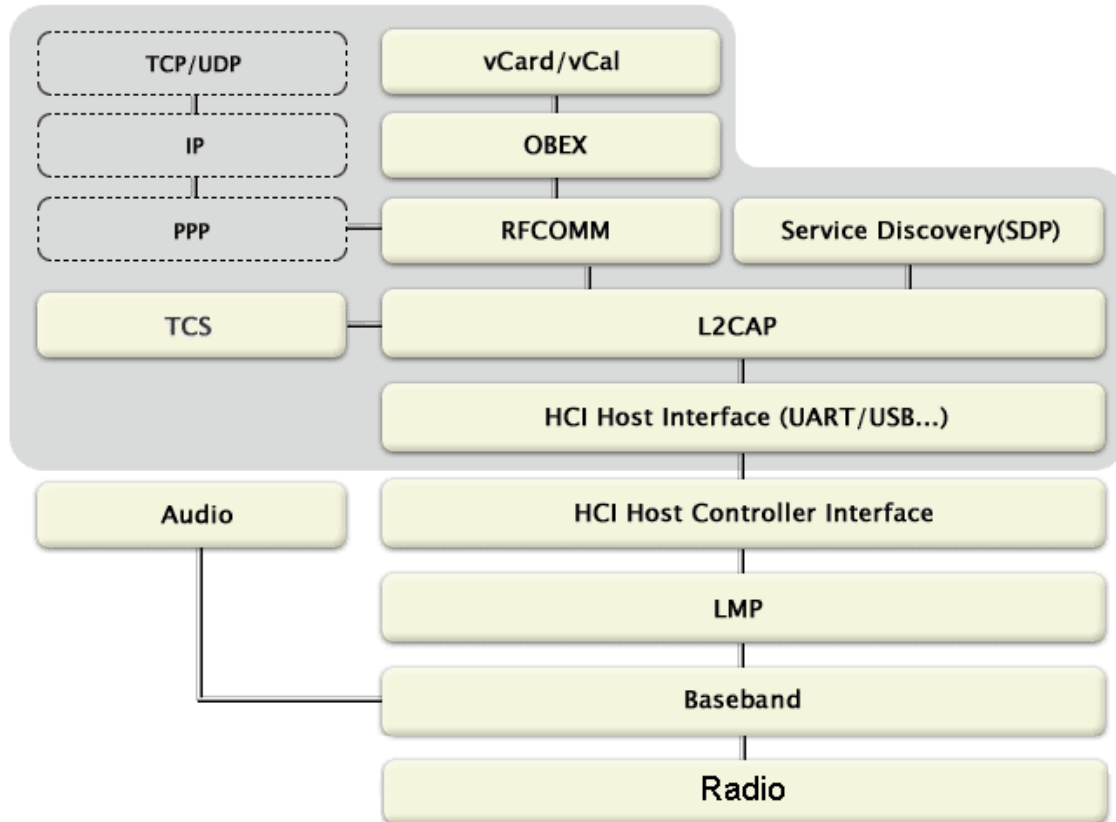
In chapter 1.2 (page 7), it was mentioned that the incentive for this thesis was the existence of a previous one. This was Christos Strydis' diploma thesis, developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete (2003) [5] and focused on BlueTooth. That thesis gave a big thrust to the continuance of the research on the BlueTooth protocol. The structure of the embedded system that has been developed in the present thesis (Bluetooth Multi-UART, BluMiU) is based on the initial foundation that was set by the architecture of the previous application platform and application that was then developed (BlueAppIE-BlueBridge), but its architecture has evolved into an entirely different than that of BlueAppIE-BlueBridge. The applications platform (BlueAppIE) has been redesigned from scratch, so that it will have improved performance and support more BlueTooth services, and the data bridge (BlueBridge) can be now considered as a fully featured data exchange mechanism which offers full duplex transfer, high data rates, client-server architecture with many clients and a simple but very effective protocol for controlling and manipulating the exchange of data. In the following sections the BlueTooth protocols and services that have been embedded in the BT module used will be presented, the architecture of the BlueAppIE-BlueBridge application environment and application will be described, its pros and cons will be shown and the architecture of bluMiU will be described. BluMiU will be specified in detail in chapter 4 (page 44).

#### **3.1. *The BlueTooth protocol***

The BT module that was used in the context of bluMiU and BlueAppIE-BlueBridge can be considered as a black box that executes instructions given to it from an external device through an interface, connects to other BlueTooth enabled devices and then can exchange data with them. However, if the parts of the BlueTooth protocol that are embedded in the BT module were known, understanding the functions of the software and hardware parts of this thesis would be rather easier and a wider knowledge on the thesis' matter would be gained. The explanation of the BlueTooth protocols is based on



BTspec 1.2. The BT module used is qualified for a different version of BTspec, but during the development of this thesis it worked perfectly, even though bluMiU is based on BTspec 1.2.



**Figure 3-1 Bluetooth Protocol Stack version 1.2 (BT spec 1.2 [3])**

The BT module is a Bluetooth Controller, BTspec 1.0b compliant, which specifies the protocol stack illustrated in Figure 3-1 (the same protocol stack is specified by BTspec 1.2). The module's characteristics are examined thoroughly in Appendix B (page 92). The layers of the Bluetooth protocol stack that are embedded in the BT module that is used are:

- The Radio, where the physical channel resides and data is transmitted between Bluetooth devices
- The Baseband, a link controller which carries out the baseband protocols and other low-level link routines [3]
- The Link Manager Protocol (LMP), which is used for link set-up and control [3] and

- The Host Controller Interface (HCI), which provides a command interface to the baseband controller and link manager, and access to configuration parameters [3]

BlueTooth protocol stack layers above HCI are embedded in the microcontroller used. In the following sections the layers that are embedded in the BT module will be described.

### 3.1.1. BlueTooth Radio

It is known back from chapter 2.3.1 (page 14) that BlueTooth operates in the 2.4GHz ISM band in the radio spectrum and makes use of a frequency hopping spread spectrum (FHSS) transceiver. The 2.4GHz frequency band is 2.4-2.4835GHz and RF channels are spaced 1MHz and are ordered in channel number  $k$  [3] as shown in the following formula: **[Frequency (MHz)] = 2402+k**, where  $k = \{0, 1 \dots, m-1\}$  and  $m=79$  (until 2004 Spain, France and Japan had  $m=23$ , because their ISM band was narrower). BT spec defines a frequency hopping rate of 1600 changes per second. A new frequency is selected by the baseband in a pseudo-random manner every 625 $\mu$ s and it is used until a next frequency is selected. The time of 625 $\mu$ s between the change is called a time-slot. By the use of frequency hopping BlueTooth ensures that interference from other devices and protocols will be kept to a minimum because the signals spread in the ISM band and it is very unlikely for two devices to interfere again after they have interfered once in a time-slot, because they probably won't jump to the same frequency during the next hop. The receiver sensitivity must be below or equal to -70dBm. The minimum output power of the BlueTooth transmitter is defined to 0dBm (1mW) for communication in a range of 10m (class 3 devices), while the maximum is between -30 and +20dBm (100mW) for 100m (class 1 devices). [3]

Symbol rate is 1Mbps with the use of a GFSK<sup>1</sup> modulator. The maximum data rate that can be achieved though is lower, because of the overhead of different protocol layers over the radio. This is 723.2Kbps for transmission, when reception is 57.6Kbps, while for a symmetric transmission-reception the maximum data rate is defined to 433.9Kbps.

---

<sup>1</sup> The signal passes through a Gaussian filter and then goes through an FSK modulator (Frequency-Shift Keying)

For full duplex transmission, a Time Division Duplex (TDD) is used. TDD is the application of Time Division Multiple Access, where the communication channel is divided into numbered time-slots and signals can only be received or transmitted in certain time-slots.[4] TDD is explained in more detail in section 3.1.2.3 (page 24).

### 3.1.2. Baseband

#### 3.1.2.1. BlueTooth topology

The BlueTooth system supports point to point connections or point to multipoint. In a point to point connection the communication channel is separated through the TDD between two BlueTooth devices. In a point to multipoint connection the same communication channel is separated through the TDD between many devices. Two or more devices that share the same communication channel form a *piconet*. Only one device can be the Master of a piconet, while all the other devices that are in the same piconet are the Master's slaves. (Figure 3-2) The maximum number of devices that can be active in a piconet is 7. [3] More than 7 devices can exist in a piconet in various sleep modes, mostly for power saving purposes. They are not active, but remain synchronized with the piconet hopping scheme and can become active without restarting the connection process. The access of the active devices in a piconet is determined only by the Master.

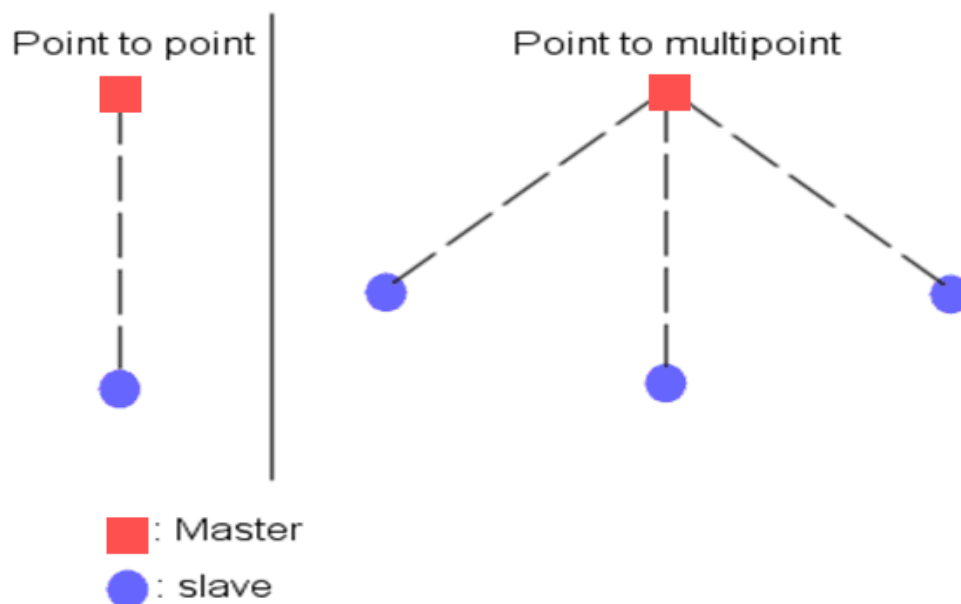


Figure 3-2 Point to point and point to multipoint connections

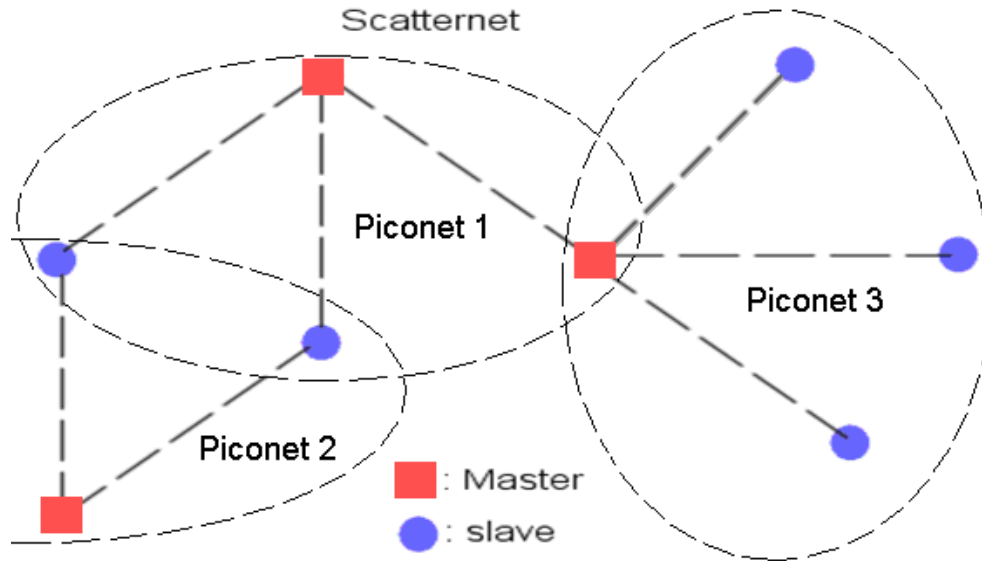


Figure 3-3 Scatternet

Piconets that have common devices are called a scatternet. (Figure 3-3) Each piconet has only one Master; however, slaves can participate in different piconets on a time-division basis, but the Bluetooth core protocols do not, and are not intended to offer such functionality, which is the responsibility of higher level protocols. [3] In addition, a Master in one piconet can be a slave in other piconets. Piconets in a scatternet are not frequency synchronized and each piconet has its own hopping sequence.

### 3.1.2.2. Master-slave roles

As seen in the previous section, in a piconet one and only one device is the Master and the rest of them are slaves. The basic piconet physical channel is defined by the Master of the piconet and the Master is the device that initiates a connection by paging. The frequency hopping in the piconet physical channel is determined by the Master's clock and BD ADDR<sup>1</sup>. When the piconet is established, the Master clock is communicated to the slaves. Then each slave adds an offset to its native clock to synchronize with the Master clock. Since the clocks are independent, the offsets must be updated regularly. All devices participating in the piconet are time-synchronized and hop-synchronized to

---

<sup>1</sup> Each Bluetooth device is characterized by this unique 48-bit device address, which is like the IP address or the MAC address, and has a 28bit clock

the channel. Once a piconet has been established, Master-slave roles may be exchanged. [3]

### 3.1.2.3. Time Division Duplex in BlueTooth

The Master controls the traffic on the piconet physical channel by a polling scheme. The basic piconet physical channel is characterized by a pseudo-random hopping through all 79 RF channels. The basic piconet physical channel is divided into time slots, each 625  $\mu$ s in length. The time slots are numbered according to the most significant 27 bits of the clock  $CLK_{28-1}$  of the piconet Master. The slot numbering ranges from 0 to  $2^{27}-1$  and is cyclic with a cycle length of  $2^{27}$ . The time slot number is denoted as  $k$ . The TDD scheme is used where master and slave alternatively transmit, as seen in Figure 3-4 below. The packet start shall be aligned with the slot start and one can extend over up to five time slots. [3]

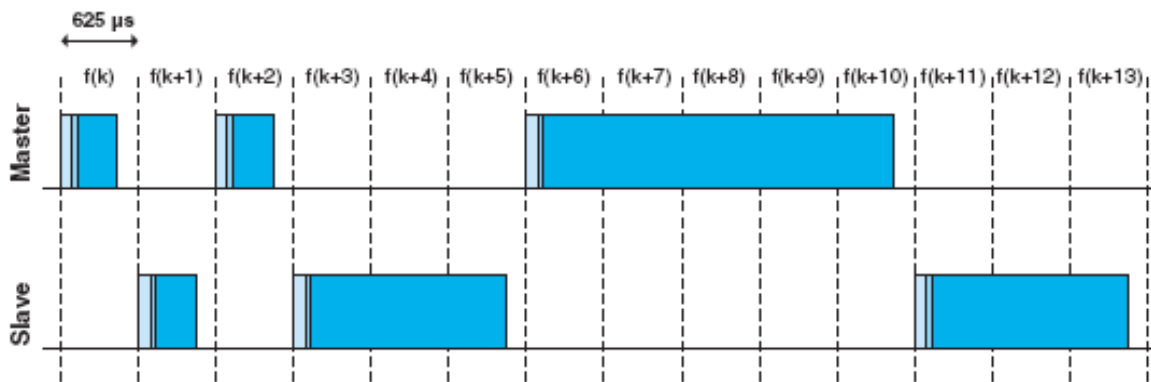


Figure 3-4 TDD in BlueTooth

### 3.1.2.4. BlueTooth connections

For a piconet (or a scatternet) to be formed, a device (each piconet's Master) must connect to the devices that will form it. A BlueTooth device has the following operational modes (states): standby, inquiry, page and connected, according to Figure 3-5. More specifically, the modes are:

- *Standby* is the default operational mode of a BlueTooth device. When a device is in this operational mode, it typically idles with only its native clock operating in a low-power mode. [1] From the Standby state the device can move to Page or Inquiry state.

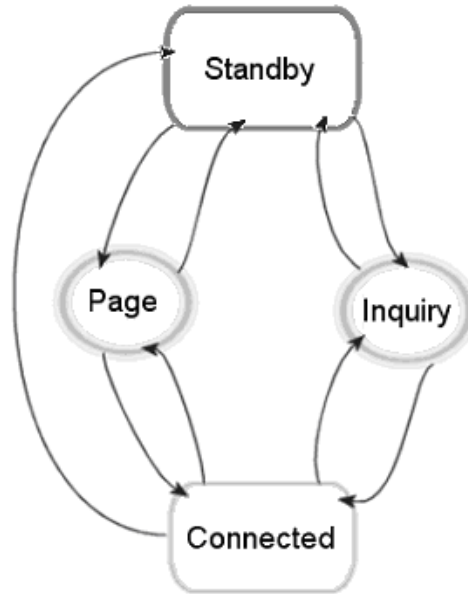


Figure 3-5 Bluetooth operational modes (states)

- *Inquiry* is the operational mode where the device that it is in this mode learns about the identity of other devices in its vicinity; these other devices must be in an *inquiry scan* state to listen for and subsequently respond to inquiries. [1]
- *Page* is the operational mode where a Bluetooth device (1) explicitly invites another Bluetooth device (2) to join the piconet whose master is (1); device (2) must be in the *page scan* state to listen for and subsequently respond to pages. [1] As Figure 3-5 shows, an inquiry by a device is not explicitly needed by a device to page another, because the identity of the device to be paged can be known to the paging device.
- *Connected* is the operational mode where a Bluetooth device is a member or the Master of a piconet. In this operational mode the device can exchange data (or voice) with the devices connected to it, disconnect with any one of them (if no devices are left connected to it, it returns to the standby operational mode) or perform inquiries and pages for additional devices to join this or some other piconet. In the latter case, a scatternet eventually would probably be created.

#### 3.1.2.5. Packet types

The general packet type is shown in Figure 3-6. Each packet consists of 3 entities: the access code, the header and the payload. In the figure, the number of bits per entity is indicated.

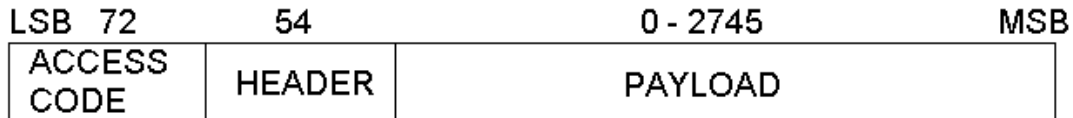


Figure 3-6 BlueTooth baseband generic packet (BTspec 1.2 [3])

Different packet types have been defined. Thus, a packet may consist of the following elements:

- A shortened (68 instead of 72 bits) access code only
- The access code and the packet header (72 and 54 bits. 126bits totally)
- The access code, the packet header and the payload (minimum  $72+54+1=127$ bits, maximum  $72+54+2745=2871$ bits)

The first two types are reserved for common packets, which are control packets essential for the BlueTooth protocol to function. Analysis of these packets is out of the scope of this thesis. The latter packet type includes the Asynchronous Connection-less (ACL) packets and the Synchronous Connection-oriented (SCO) packets.

The SCO packets are typically used for 64kbps speech transmission and this thesis does not deal with them.

The ACL packets are used for asynchronous logical transport and the payload can be user or control data. [3] They can cover from 1, 3 or 5 time slots and they provide a 16bit Cyclic Redundancy Check (CRC) code. There are Data - Medium Rate (DM) and Data - High Rate (DH) ACL packets. Thus, there are DM1, DH1, DM3, DH3, DM5 and DH5 packets, with the number denoting how many time slots they occupy. The only difference between them DM and DH is that DM ACL packets provide the information plus the CRC code coded with a rate 2/3 Forward Error Correction (FEC).

### 3.1.3. Link Manager Protocol

The Link Manager Protocol (LMP) is used to control and negotiate all aspects of the operation of the Bluetooth connection between two devices. This includes the set-up and control of logical transports and logical links, and the control of physical links. [3] It is used to communicate between the Link Managers (LM) on two devices which are connected for ACL logical transport. The LM provides

1. Security Management, which provides device authentication and encryption
2. Power Management, which regulates the device's association with the piconet it's connected, so that it would preserve power. [1] There are three modes that

can be used to reduce power consumption: sniff, hold and park (examined below)

3. QoS Management, which regulates the bandwidth used in connections
4. Connection Management, which manages the paging parameters, the Master-slave roles, the clock of the BlueTooth devices and the connection establishment and link detachment

These services are provided by the LM by the use of LMP, which exchanges LMP messages on connected BlueTooth devices. All LMP messages apply solely to the physical link and associated logical links and logical transports between the sending and receiving devices. The protocol is made up of a series of messages which are transferred over the ACL-C logical link, which is a control link used by the LM, resides on the default ACL logical transport between two devices, uses DM1 packets and has a higher priority than other traffic. [3] LMP messages are interpreted and acted-upon by the LM and are not directly propagated to higher protocol layers.

The most common and flexible methods for reducing power consumption are the use of sniff and park. Hold can also be used by repeated negotiation of hold periods. [3]

### **3.1.3.1. Sniff mode**

In sniff mode, the duty cycle of the slave's activity in the piconet may be reduced. If a slave is in active mode on an ACL logical transport, it must listen in every ACL slot to the Master traffic, unless that link is being treated as a scatternet link or is absent due to hold mode (explained below). With sniff mode, the time slots when a slave is listening are reduced, so the Master only transmits to a slave in specified time slots. The slave listens in Master-to-slave transmission slots starting at the sniff anchor point. The sniff anchor points are spaced regularly with an interval of  $T_{sniff}$ . (Figure 3-7) To enter sniff mode, the Master or slave issue a sniff command via the LM protocol. This message includes the sniff interval  $T_{sniff}$ . [3]



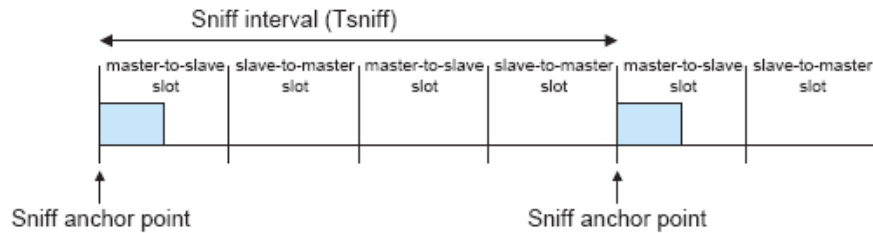


Figure 3-7 Sniff Anchor Points (BTspec 1.2[3])

### 3.1.3.2. Hold mode

During the connection state, the ACL logical transport to a slave can be put in a hold mode. In hold mode the slave temporarily doesn't accept ACL packets on the channel. With the hold mode, the device is free to do other things like scanning, paging, inquiring, attending another piconet, or entering a low-power sleep mode. Prior to entering hold mode, Master and slave agree on the time duration the slave remains in hold mode. After the end of the agreed time, the slave wakes up, synchronizes to the traffic on the channel and waits for further Master transmissions.

### 3.1.3.3. Park state

When a slave does not need to participate on the piconet channel, but still needs to remain synchronized to the channel, it can enter park state. Park state is a state with very little activity in the slave. All messages sent to the parked slaves are carried by broadcast packets (packets broadcasted to multiple devices). The parked slave wakes up at regular intervals to listen to the channel in order to re-synchronize and to check for broadcast messages. With this method, the maximum reduction in the device's power-consumption can be achieved.

### 3.1.4. Host Controller Interface

A Bluetooth Controller contains the Bluetooth Radio, the Baseband, the LM, a resource controller and a device manager. These parts of the Bluetooth protocol are used by a Bluetooth Host which executes other higher level protocols. The Bluetooth Controller and the Bluetooth Host communicate through the Host Controller Interface (HCI), which is defined by the Bluetooth SIG as the physical interface along with a transaction-style communication protocol to carry information between the Host and the Controller. (Figure 3-8) The main goal of this transport layer is transparency. The Host

Controller driver (which interfaces with the Controller) is independent of the underlying transport technology. This allows the HCI to be upgraded without affecting the transport layer.

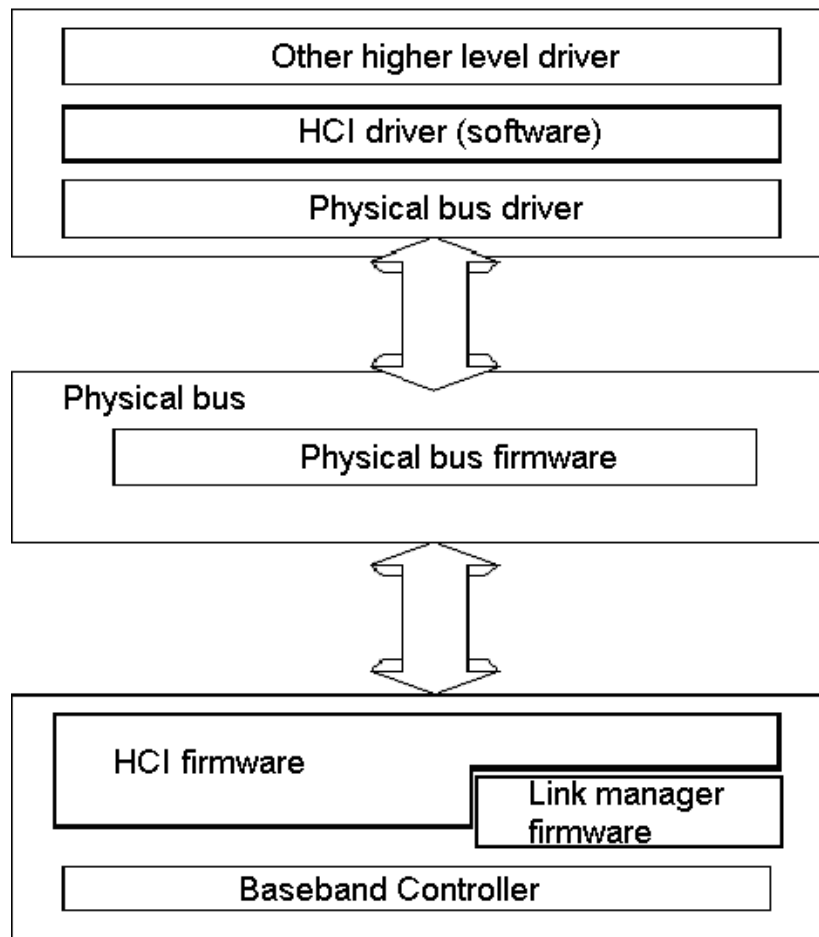
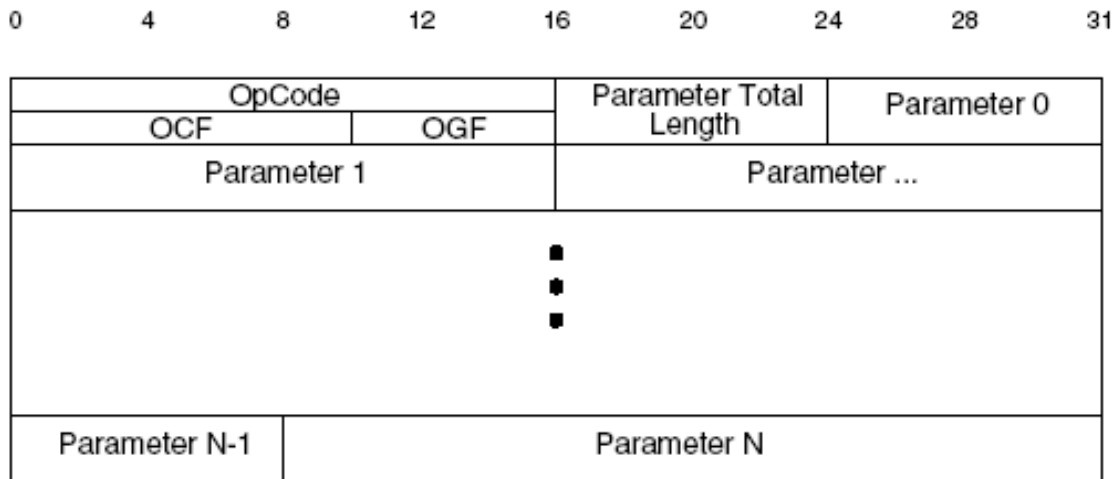


Figure 3-8 The HCI transport layer (BTspec 1.2 [3])

The traffic crossing the HCI is: the command packets, the event packets and the data packets.

The Host issues the HCI commands and can access through them all functions of the Bluetooth Controller such as setting operational parameters, configuring the module's operational status, reading and writing specific low-level registers. The format of HCI commands is shown in Figure 3-9. The HCI portion of the BT spec is the largest one, since only by the use of HCI commands a device can communicate with the lower layers of the Bluetooth protocol in the Bluetooth Controller. Each command is

assigned a 2 byte Opcode used to uniquely identify different types of commands. The Opcode parameter is divided into two fields, called the Opcode Group Field (OGF) and Opcode Command Field (OCF). The OGF occupies the upper 6 bits of the Opcode, while the OCF occupies the remaining 10 bits. [3] Then a byte indicating the length of the parameters that will follow, as well as their number, must exist since every parameter is one byte long and after that all the parameters follow.

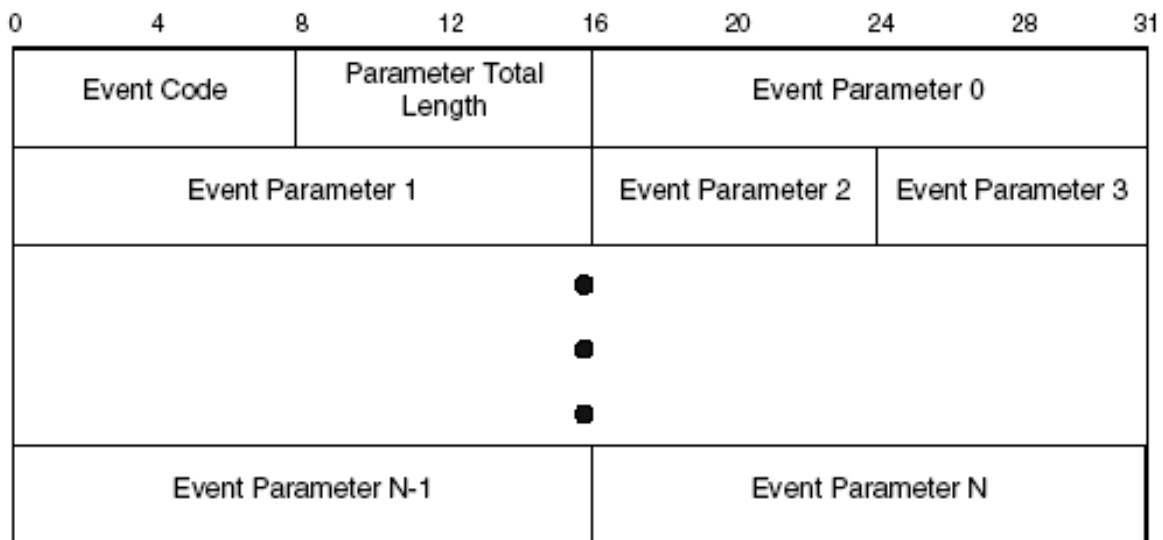


**Figure 3-9 HCI command format**

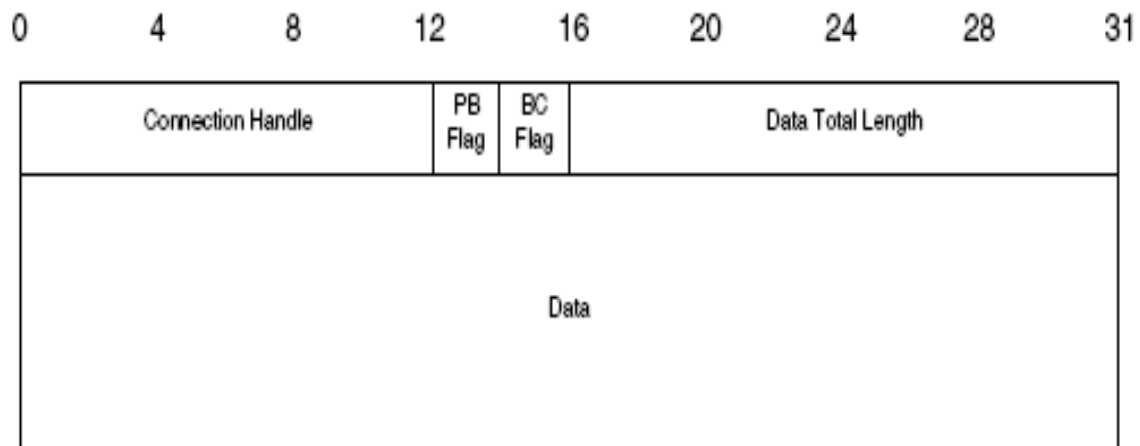
The Controller notifies the Bluetooth Host of the outcome of a command or of an event that took place in a device connected to the Controller with an HCI event. The format of the HCI event packets is shown in Figure 3-10. Each event is assigned a one byte event code used to uniquely identify different types of events. Then parameters follow in the same way as in HCI commands.

Data between Bluetooth devices is exchanged through the HCI layer by the use of ACL data packets. Their format is shown in Figure 3-11. In the beginning of the ACL packet, there are 12bits for the connection handle of the device to which the data will be sent to. The two bits that follow indicate if this packet is the first packet of a higher layer message (0b10) or a continuing fragment packet of a higher level message (0b01). Then two more bits are next that indicate if the packet is a point-to-point packet (0b00), a packet that will be sent or was sent to all the slaves (0b01) or a packet that will be sent or was sent to all the slaves that are in the park, sniff or hold mode (0b10). Then 2

bytes indicating the length of the data that is contained in this ACL packet will follow and, finally, the data itself.



**Figure 3-10 HCI event format**



**Figure 3-11 ACL data packet format**

The Host transmits to the BlueTooth Controller an ACL data packet through the HCI and then the Controller wirelessly transmits the ACL packet to the target BlueTooth Controller. Then by the reverse course the ACL packet arrives to the other end's Host.

The supported by BT spec 1.2 HCI transports are 3-wire, SD-transport (Secure Digital), UART and USB. In this thesis the UART transport is used for communication with the BlueTooth Host, which in this case is the microcontroller. In the Host that was designed, the Bluetooth Logical Link Control and Adaptation Protocol (L2CAP) is implemented and

higher level protocol multiplexing, packet segmentation and reassembly is supported through the very effective Serial Port Profile<sup>1</sup> (SPP) that was designed.

In the sections below an overview of the existing system architecture will be presented.

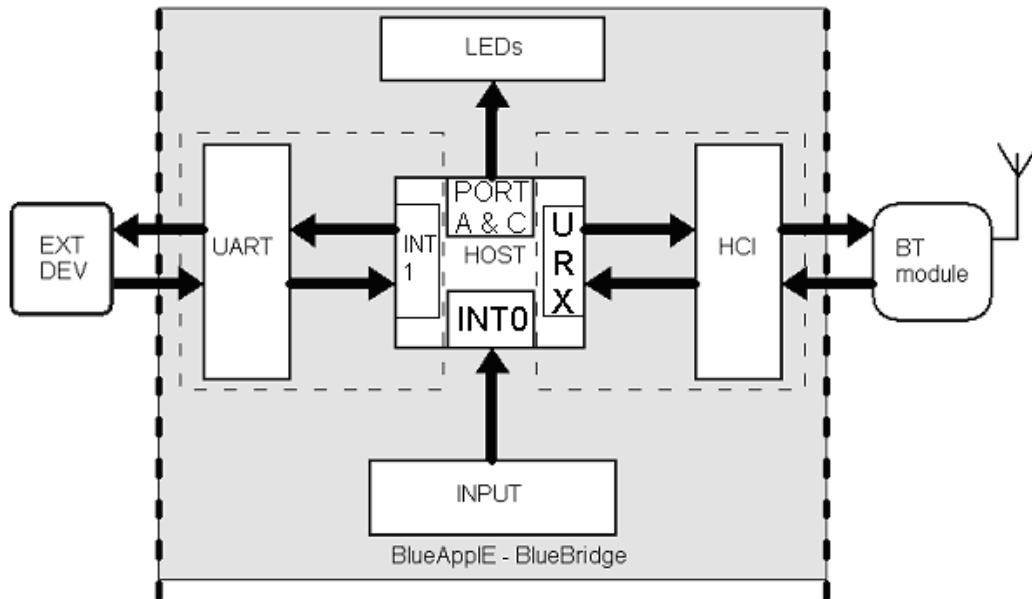


Figure 3-12 BlueApple - BlueBridge

### 3.2. The existing system (BlueApple-BlueBridge)

BlueApple-BlueBridge is an applications environment, developed at the Microprocessor and Hardware Laboratory (MHL), as part of the diploma thesis by Christos Strydis [5], of BlueTooth (BlueApple) (which was designed to fully control and exploit the capabilities of a BlueTooth module) that can setup and run a wireless transparent UART bridge (BlueBridge) [5] application for transferring data between two BlueApple-BlueBridge devices and in extent to any two devices that have been connected to each one of the BlueApple-BlueBridge devices. The most important component of this system is a

---

<sup>1</sup> The SPP that was designed is compatible with BlueTooth devices supporting this specific SPP and *not* the BlueTooth SIG SPP, because it demands memory and processing resources not available in a microcontroller like that used

microcontroller (HOST) that is used to control all the other components that BlueApple-BlueBridge consists of. These components are the user input (INPUT), the communication (HCI) with the BT module, the communication (UART) with the external devices (EXT DEV) and the output led matrix (LEDs). (Figure 3-12) Through the BlueApple-BlueBridge description its advantages and disadvantages will be stressed out.

### 3.2.1. Hardware

The components that BlueApple-BlueBridge consists of, which were mentioned previously, are the following:

- I. **BlueTooth module (BT module):** The same module that is used in bluMiU is used, that is described in more detail in Appendix B (page 92). It is a BTspec 1.0b qualified BlueTooth Controller and embeds the following layers of the BlueTooth protocol stack that were described in the previous sections: the BlueTooth Radio, the Baseband, the Link Manager, a UART and a USB HCI. It also has a short range and low-power internal antenna. In the BlueApple (as well as the bluMiU) UART is used for the communication of the BT module and the Host; UART is also used for the communication of the Host and the external device.
- II. **Host:** Since the BT module used did not have the amount of necessary CPU power to embed the entire BlueTooth protocol, a device with an efficient amount of processing power was needed to execute the upper BlueTooth protocol layers. For this purpose the Atmel AVR AT90S8515 microcontroller was used. The use of the specific microcontroller was not a random choice. The following reasons made this microcontroller a good choice at that time:
  - ✓ Availability
  - ✓ Reliability
  - ✓ Low cost
  - ✓ Easy to use and program

These advantages, in conjunction with the fact that it seemed to cover the BT module in processing power, gave the AVR AT90S8515 the privileged Host position, while certain disadvantages this microcontroller had were overlooked at that time.

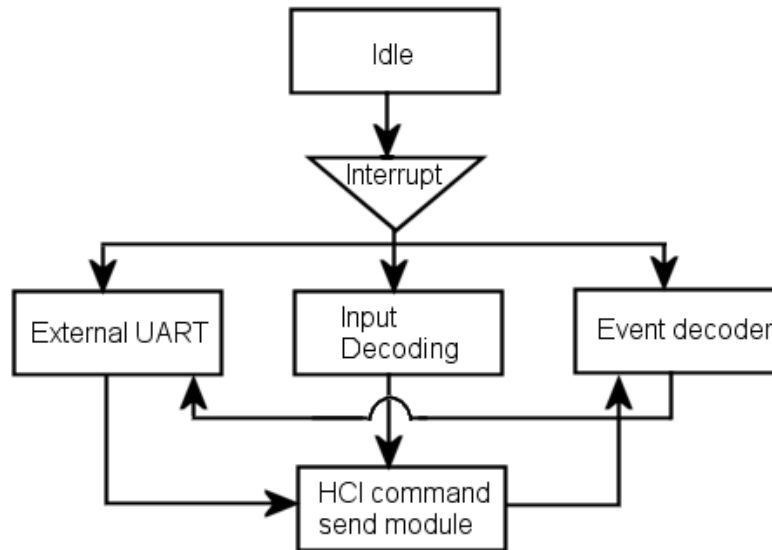
- III. **Input & LEDs:** The input system consists of 8 push-buttons that their signals are encoded from 8 signals to 3 through a 74F148 priority encoder. These 3 signals end up to the Host. The 8 push-buttons control all the functions of the Host and the BT module. For the display of the various Host-BT module functions, as well as the BT module outputs, a matrix of 16 LEDs, separated in two groups with 8 LEDs each, is used. The first group of LEDs displays the selected command or device, depending on the selected mode, and displays error messages on the reception or transmission of packets. The second group displays error messages or the status of the system.
- IV. **External UART:** The BlueApple-BlueBridge design needed an extra UART, in addition to the one the microcontroller (mC) had that was used for the HCI transport between the Host and the BT module, for the BlueBridge application. For this purpose, instead of selecting another mC with two UARTs, something that would mean virtually that BlueApple-BlueBridge development should be started over from the beginning, the design adopted an external UART solution provided by MAXIM, the MAX3110E. This solution, though, needs software resources from the mC, which are limited, and, thus, the MAX3110E can't achieve the data rates it supports. This, in addition to the fact that no data buffering mechanism has been implemented, doesn't allow the BlueApple-BlueBridge system to reach acceptable data rates.

### 3.2.2. Software

BlueApple-BlueBridge software architecture was divided in two parts. The *BlueApple* which, as seen previously, is a Bluetooth application environment, that controls the BT module's functions through the UART HCI layer, and the *BlueBridge* which is an application for the BlueApple that provides wireless UART communication. The main task of the BlueApple-BlueBridge design is to issue Bluetooth HCI commands to the BT module and correctly decode the events they return after they have been executed by the module.

#### 3.2.2.1. BlueApple software architecture

A generalized flow chart of BlueApple's software architecture and operation can be seen in Figure 3-13 and an explanation of its functions follows.



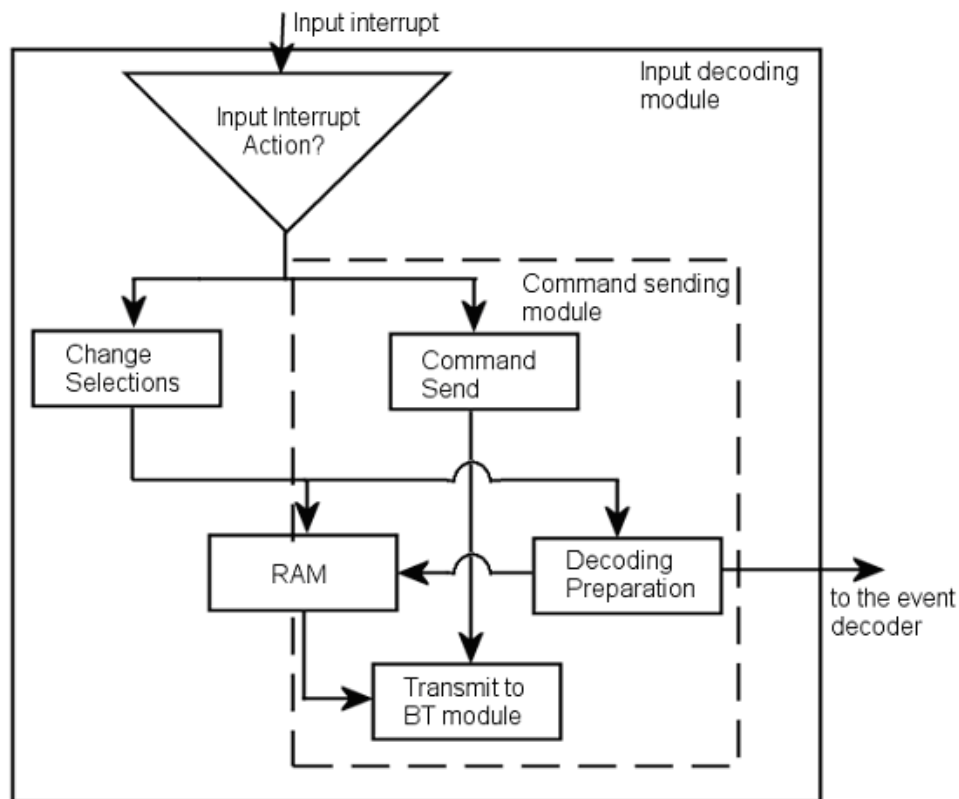
**Figure 3-13 BlueApple operation flow chart**

The idle process puts the mC in a power saving mode until an interrupt is triggered. When the software detects a supported interrupt, it leads the mC to execute the appropriate module. After the end of each of the modules' jobs, the mC returns to the idle process. Interrupts originate from the input, from internal UART or from the external UART. The modules dedicated to every one are:

- The input decoder takes actions depending on the button pushed. A flow chart of the input decoder and the command sending module is depicted in Figure 3-14. There are 3 kinds of input interrupts:
  1. Making a selection. They change various pointers of the system, which point on certain locations of the mC's RAM. These pointers either select a command that the user will later transmit to the BT module, or a remote device that the selected command will be targeted to. The selections are shown on the LEDs, following a binary count.
  2. Commanding the mC to transmit a command to the BT module. In this case, the number of the command and all the selections made are forwarded to the command sending process.
  3. Running the BlueBridge application. The appropriate module, specified in the following section, is executed. This allows data transfers between 2 connected BlueBridge devices.



- The tasks of the command sending process (Figure 3-14) are to transmit HCI commands or ACL packets to the BT module and to properly prepare the event decoder for accepting incoming event packets from the BT module. When it is executed, it reads from the memory the byte stream of the selected command and the characteristics of the selected device (if they are needed) and then uses the mC's internal UART to transmit the bytes read. After commands are executed by the BT module, a Bluetooth event is usually returned to the mC. Depending on the command sent, the command sending module notifies the event decoder of the byte sequence of the expected event and, thus, prepares the event decoder so that it will function correctly.



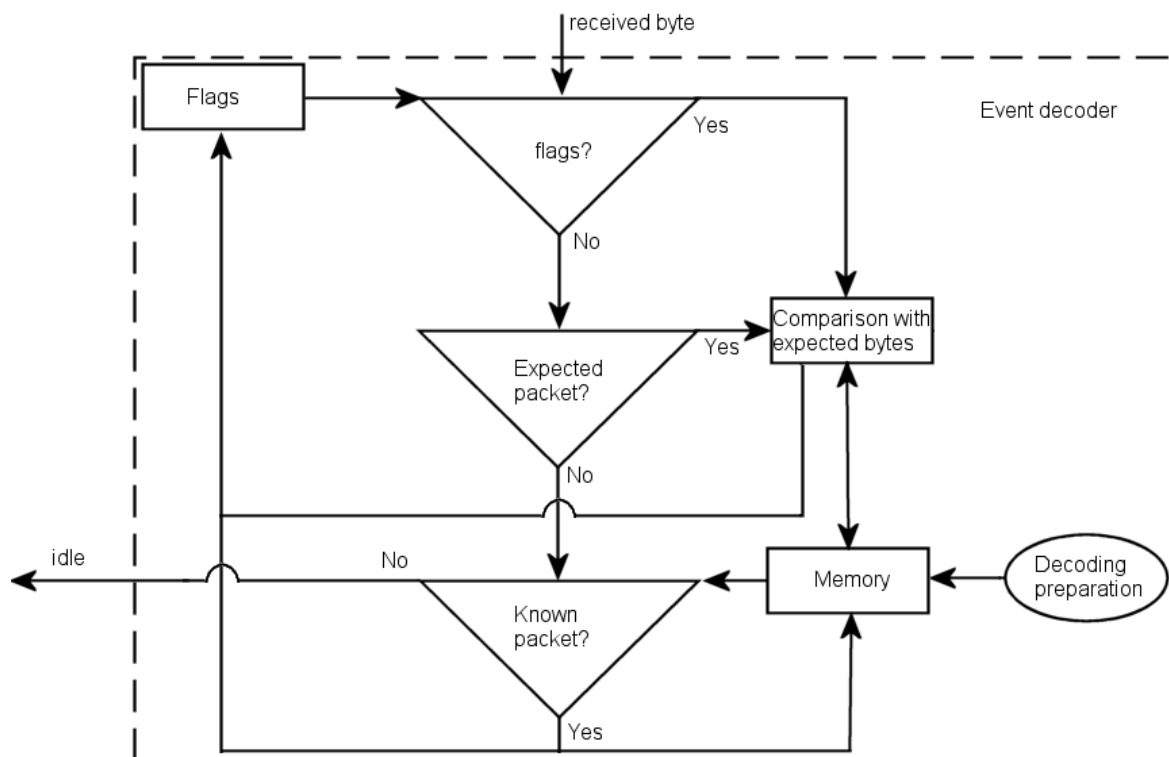
**Figure 3-14 BlueApple input decoding module & command sending module**

- The previously mentioned event decoder idles until a byte is received from the mC's UART. The data flow chart of the decoder is depicted in Figure 3-15. When a byte arrives:

1. If flags have been set the byte is compared with the expected bytes generated by the command sending module in the decoding preparation module. More flags

are set, LEDs are lit and data is written to the memory. Then the decoder waits for more bytes to arrive from the UART until the packet has been fully decoded.

2. If flags have not been set and the decoder is prepared to receive an event, the header of the packet being received is checked if it matches the expected event. If it is the one expected, the appropriate flags are set and more bytes received are decoded normally according to scheme (1). If it is not the expected event the decoder checks if it matches one of the known remotely triggered events (data packets, create connection request packets, disconnection packets). In case the packet being received is a known packet, its byte stream will be compared according to scheme (1) with a template of this packet stored in the memory.
3. If the header cannot be recognized to be a known packet and nothing is being expected, the received packet is ignored.



**Figure 3-15 BlueApple event decoding module**

BlueTooth commands implemented in BlueAppleE are stored in the mC's memory in two copies: one in the RAM of the mC and one in the non-volatile flash memory (FLASH) it has for storing the program code. When the mC is powered up, the commands stored in

the FLASH are transferred to the RAM. This method cannot be, by any means, characterized as memory efficient, since the mC used has *only* 512Bytes of memory.

As a conclusion, the BlueAppleE has some disadvantages, which were quite serious:

- No buffering mechanism was implemented
- The command sending module was not memory efficient
- The event decoding module was not memory efficient, and depended on the last command that was sent by the BlueAppleE
- The decoding preparation module would be useless, if a better event decoding module was implemented

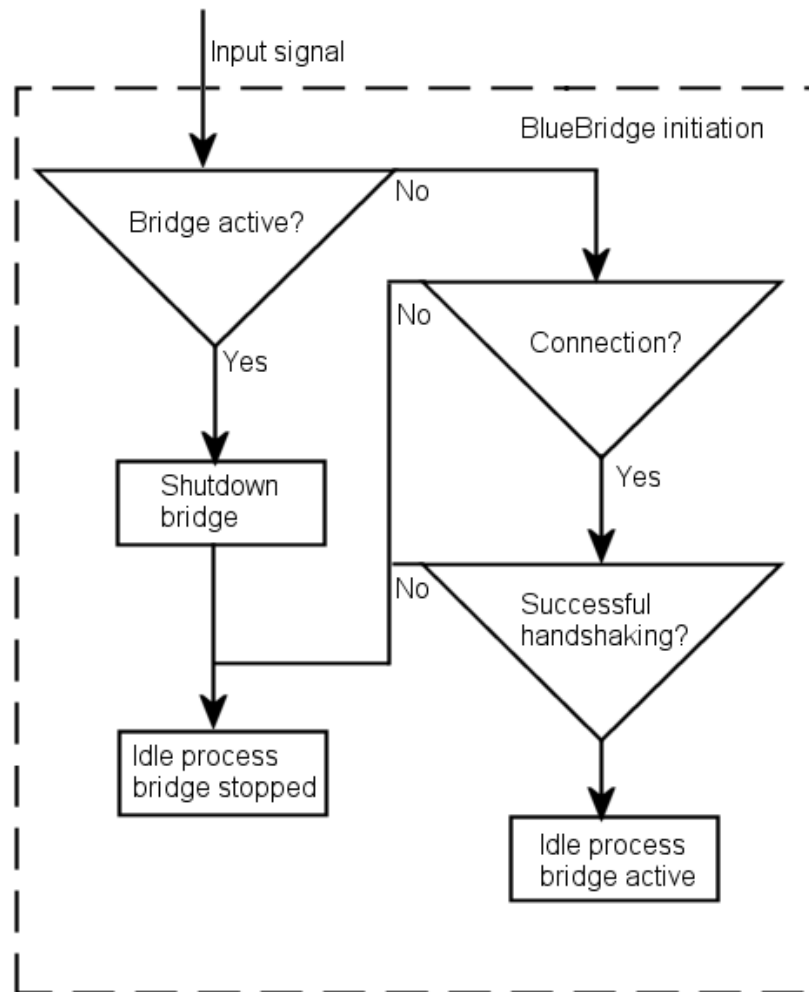
### 3.2.2.2. BlueBridge software architecture

The BlueBridge application provides a way for two devices connected to the external UARTs of two BlueAppleE-BlueBridge platforms to exchange data. This application uses the command sending and event decoding modules of the BlueAppleE to function. It keeps in the mC's memory the byte sequence of a data packet's template.

For the BlueBridge to work both ends of two connected BlueAppleE devices must start the BlueBridge application. The handshaking process used is depicted in Figure 3-16. Through the input the user requests a data bridge to be initiated. If another bridge is running, it is stopped. If a bridge is not running and a connection is present, the BlueBridge application sends a control signal to the remote device, informing it that it will start a data bridge with it. If the remote device accepts, data received from the external UART of the BlueAppleE will be encapsulated to data packets and transmitted to the remote device. Of course data packets received by that device will be decoded, the data they contain will be de-capsulated and then transmitted via the external UART.

Only the two devices that started the BlueBridge can exchange data after the handshaking that has been committed by the initiation of the application, making BlueBridge a connection-oriented service. Connection-oriented services are used for complex systems which require control messages to be exchanged, in order to prepare for an onslaught of packets, and may demand *reliable data transfer* (through an unreliable communication channel) and/or *flow* and *congestion control*. [13] *Reliable*

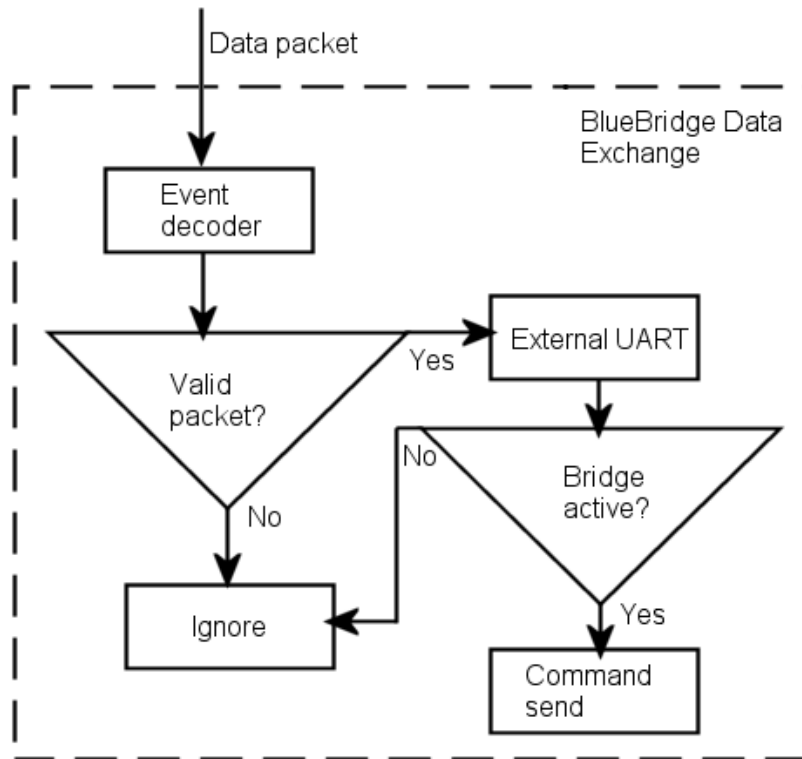
*data transfer* is when the application can rely on the connection to deliver all its data without error and in the proper order. *Flow control* makes sure neither side of a connection overwhelms the other side by sending too many packets too fast, while *congestion control* helps preventing the network from entering a state of gridlock, which causes loss of packets. [13] Though, the mechanism for data transfers provided by the Bluetooth protocol (ACL) is reliable, simple and targets point-to-point (or multipoint) transfers. There is no need to design a connection-oriented service when dealing with a simple cable replacement protocol, such as Bluetooth.



**Figure 3-16 BlueBridge Initiation**

The decoding of data packets is done using the same mechanism that BlueAppleE uses to decode events. (Figure 3-17) The expected byte stream will be a data packet template. After the expected bytes are checked for validity, the payload byte is forwarded to the higher level application through the external UART. This means that

the header bytes (9 bytes) are practically ignored<sup>1</sup> and the payload byte is forwarded to a higher level application through the external UART. If a data bridge is active, every byte received from the external UART will be forwarded to the command sending module that will encapsulate it into a data packet (by adding to it the 9 byte header) and then transmit it to the BT module.



**Figure 3-17 BlueBridge data exchange through the BlueApple**

As seen, in every 10 bytes sent over the air by the BlueBridge application, only one is useful. This automatically reduces the effective data rate from 57.6kbps<sup>2</sup> to 5760bps, a significant reduction in the bandwidth that is further reduced by the fact that the UART is external. Concluding the description of the existing system's architecture and functionality, the BlueBridge application also has certain disadvantages, in addition to those of BlueApple:

---

<sup>1</sup> They are not completely ignored; the header is checked to determine if the source BlueTooth device is the one with which the BlueBridge application has started, if the all its elements are correct and if the payload byte is only one

<sup>2</sup> The default data rate of the BT module

- Too low data rate (1200bps)
- Connection oriented operation, when it's not needed
- Only *one byte* data payload in every *10 bytes* transmitted over the air
- Supports only one ACL connection

### 3.3. ***The new system: BluMiU architecture overview***

BlueTooth Multi-UART (bluMiU), developed at the Microprocessor and Hardware Laboratory and designed in this thesis, had some *basic* goals to achieve. These were:

- To improve the performance of BlueBridge,
- Overcome its disadvantages (noted in the previous section) and
- Transform it to a reliable and robust data exchange BlueTooth enabled system

Once these requirements were met, *new* services would be added. These requirements, as was found in a project to study BlueBridge and improve its data transfer rate, could only be achieved by:

- Changing the mC that was used, so that it could cover the requirements. The AT80S8515 was changed by the ATmega161 AVR. The reasons that led to this change are discussed in the next chapter.
- Implementing a buffering mechanism in its UARTs, so that efficient data exchange and multiprogramming could be achieved.
- Replacing *every* software module that was embedded in the AVR, because more efficient modules were needed.
- Adopting a client - server architecture, so that true point-to-multipoint connections could be achieved.

Multiple data connections are supported by the server and a simple, yet very effective, protocol for data communication with every connected device is provided. The *only* module that was changed (a.k.a. not replaced completely) was the input module that controls the decoding of the push-buttons' signals. A brief list of the differences between BlueAppleE-BlueBridge and bluMiU and the new services the latter introduced was seen at the chapter 1.2 (page 7).

BlueAppleE didn't have an exact focus. It was more like an attempt to discover the capabilities of the Bluetooth protocol and the BT modules by issuing to it simple commands and correctly decoding their replies, so that further research could be conducted later on. From the beginning of bluMiU development, the primary focus of the work was given on data transfers between bluMiU devices. So, mC specific functions and/or Bluetooth services that could enhance data transfers were exploited. This fact implied that the implemented Bluetooth commands and Bluetooth services contained in the BlueAppleE had to be reviewed (and new should be found) so that only the necessary elements for achieving data transfers between Bluetooth devices would remain in bluMiU. The implemented Bluetooth commands were reduced from 27 to 9 in the server; 11 in the client. The commands necessary for the discovery, the identification, the connection and disconnection of Bluetooth enabled devices in the vicinity of the BT module were the commands that remained in the bluMiU. The data transfers were mainly helped by:

1. The implementation of a buffering mechanism in the data transports. This gave the major boost in the evolution of bluMiU; relatively large amounts of data could be stacked in the buffer, waiting to be transmitted, allowing the bluMiU control unit to attend to other tasks.
2. The differentiated decoding of incoming events. BlueAppleE loaded the pattern of the expected packet to be received into the memory and then waited until this packet was received. BluMiU decodes whatever, whenever it is being received without unnecessary memory transfers, since all the possible packets that can be received have been included in the event decoder's piece of code. This is a fair solution since the mC doesn't have a quantity of RAM available to be wasted for the decoding of events and it is generally faster to read directly from the program code than from the RAM.
3. The design of a packet exchange mechanism. In BlueBridge every byte received was transmitted to the remote device. This, as seen, limits the effective data rate to the 1/10 of the available. The bluMiU design provides data packets of variable payloads that contain (except from the data payload) information for the source remote device of the packet, in order to support multiple connections. The

payload of data is limited only by the BT modules used, the BlueTooth protocol itself and the number 65536.

Having in mind the limitations in resources and the complexity of the work that the mC would have in dealing with so many things simultaneously, even in trivial tasks, great attention was given to the software parts of the system. Every module of the system was written in AVR assembly. Software architecture is based on interrupts that, when triggered, cause the appropriate Interrupt Service Routines (ISRs) to be executed. In these ISRs the modules of the system have been built. Whenever the mC is executing an ISR and a subsequent interrupt is triggered, data vital for the ISR running is stored in the mC's RAM, so that the mC can execute the ISR for the new interrupt and when it ends the first ISR will be resumed. This fact allows the mC to execute all the interrupts that occur, internal and external, without losing any data or missing any interrupts and, thus, decreasing the HOST's reliability to the external device.



## 4. System organization

---

In the previous chapters, an overview of the lower levels of the BlueTooth stack and the existing BlueTooth enabled system developed in MHL in the context of Chrystos Strydis' thesis (BlueApplE-BlueBridge) [5] was presented. We explored some of the disadvantages the BlueApplE-BlueBridge system possessed and saw what needed to be improved. The new system (bluMiU) reduces and in some cases it even eliminates the flaws of the existing system.

BluMiU offers the following characteristics that make it better than the existing one:

1. packet sending mechanism
2. client-server architecture
3. support for multiple device connections
4. connectionless data exchanging between connected clients and the server
5. payload of data packets has been significantly increased, limited by the buffer of the BT module or 65536bytes ( $2^{16}$ )
6. interrupt based multiprogramming
7. the host can be fully controlled from the device connected to the platform
8. the device connected to the platform can give commands to the BT module
9. improved data and command sending mechanism to the BT module,
10. improved decoding of incoming event and data packets from the BT module mechanism
11. improved mechanism of data exchange between a device connected to the platform and the platform itself
12. the platform reports the connected device of the addresses, the handles and the user friendly names of connected and inquired devices and
13. the platform notifies the connected device whenever a disconnection occurs

In this chapter, we will present the bluMiU hardware and software design, which was developed in this thesis and which features:

- The hardware that is used to control the BT module, communicate with it and provide Bluetooth connectivity to an external device
- The whole software design that contains the upper layers of the Bluetooth protocol stack and the data exchange protocol

In the thesis' context, two bluMiU platforms have been implemented; one server and one client. The hardware and software of the bluMiU system will be described in the following sections. At first the data communication protocol that is used in bluMiU will be examined.

### ***4.1. BluMiU data transfer protocol***

Having two (or more) Bluetooth enabled systems that can't exchange data at all, or even exchange without an acceptable data rate, is like not having them. The exact protocol, which enables bluMiU systems to exchange data between each other, will be specified in the following sections. The next section explains why the specified by the Bluetooth SIG Serial Port Profile (SPP) was not implemented.

#### **4.1.1. The Bluetooth protocol approach**

As seen in previous chapters, the Bluetooth SIG has specified a standard for the serial data transfers between Bluetooth enabled devices, the SPP (Serial Port Profile). This profile covers the scenario of setting up virtual serial ports (or equivalent) on two devices (e.g. PCs) and connecting these with Bluetooth, to emulate a serial cable between the two devices.

Any application may be run on either device, using the virtual serial port as if there was a real serial cable connecting the two devices (with RS232 control signaling). The profile assumes that the applications on both sides are typically legacy applications, able and wanting to communicate over a serial cable (which in this case is emulated). But typical applications cannot know about Bluetooth procedures for setting up emulated serial cables, which is why they need help from some sort of Bluetooth-aware helper application on both sides. Of course, according to the Bluetooth Profiles Book version 1.1, only one connection at a time is dealt within this profile. [7] Consequently, only point-to-point configurations are considered.

As with all the Bluetooth profiles, service discovery procedures and other complicated link and setup (of the virtual serial port) procedures have to be performed to set up an emulated serial cable connection. This approach is optimized so that a device, which fully supports the SPP, to cover all the interoperability requirements of all Bluetooth protocol stack layers that get involved with SPP.

### 4.1.2. The bluMiU protocol approach

The bluMiU system doesn't have the hardware and software resources to support the SPP that Bluetooth SIG has specified. So, a different protocol had to be implemented that would be suitable for the requirements of bluMiU. The bluMiU protocol that was implemented needs the external devices connected to the bluMiU to know that it needs packets to be sent to the bluMiU device that include a special header and a data payload. This header has an one byte packet indicator (0x01) that identifies the packet transmitted would be a data one, 2 bytes with the handle of the target device (the handles of the devices wirelessly connected to bluMiU are reported to the external device) and 2 bytes indicating the length of the payload that will follow (referred to as the initial length below), in a little endian byte order. After the header the data shall follow. BluMiU supports payloads between 1 and 65536 bytes (this number is further reduced by the capabilities of the BT module used). The packet format is depicted at Figure 4-1.

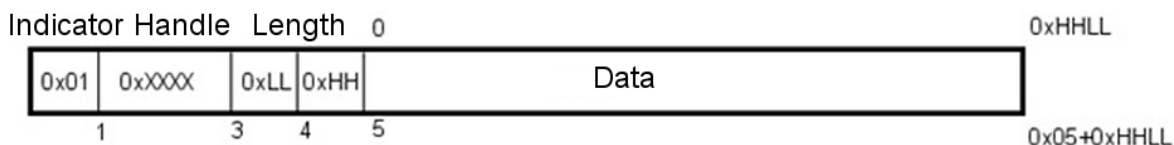


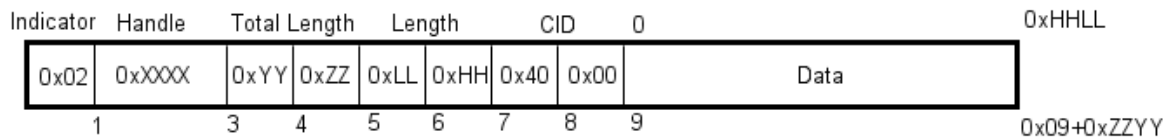
Figure 4-1 BluMiU packet

A decoder will check to see if the header is valid (the validity of the handle will be checked by the BT module). If the header is valid, the decoder will transcribe the packet received to an ACL data packet (its format is depicted in Figure 4-2, which is the format of the ACL packet that the BT module uses). The ACL packet that this BT module uses has the following fields in its header:

1. The ACL data packet indicator. [1 byte] This is a byte with the hexadecimal value 0x02.
2. Handle. [2 bytes] This is the handle indicating the device to which the data packet will be sent to. The two last bits of the handle (15 and 16) determine

whether that packet will be broadcast<sup>1</sup> to all the active and/or parked slaves, or was broadcasted by the Master to all its active and/or parked slaves.

3. Total Length (L2CAP length). [2 bytes] It is the Logical Link Control & Adaptation Protocol (L2CAP) specific length, which includes the data payload length and some other L2CAP header fields' length. The BT module used does not implement the L2CAP layer (which is an upper level Bluetooth protocol stack layer), but it uses its headers to promote interoperability. The total length is found by adding 4 to the initial length.
4. Length. [2 bytes] This is the initial length of the data included in the packet.
5. Channel ID (CID). [2 bytes] The CID identifies the destination channel endpoint of the packet. Different applications use different CIDs to retrieve data from the Bluetooth device. CIDs are something like the port numbers in the TCP/IP. In the bluMiU the CID 0x0040 is used, which is not reserved for any Bluetooth profile.



**Figure 4-2 BT module acceptable ACL packet used by bluMiU**

Every number used in the header is in a little endian byte order. After the bluMiU data packet has started to be transcribed into an ACL packet, the BT module will send it to the device indicated by the handle in it, if that device is connected to the BT module.

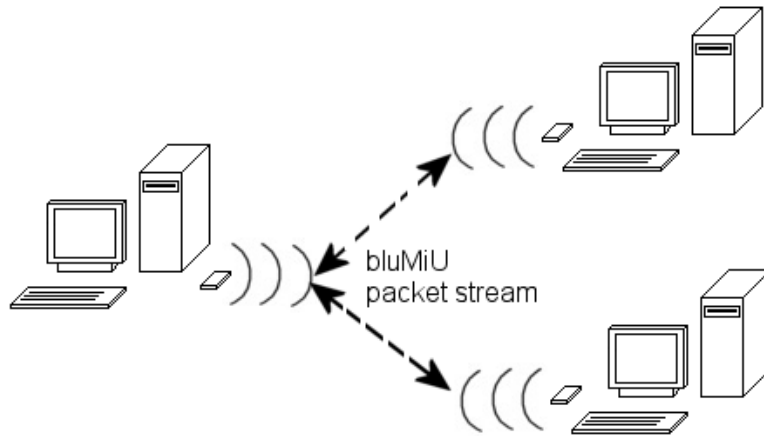
If the server wirelessly transmits an ACL packet to a client, the client's BT module receives the packet and forwards it to the host. The client shall then de-capsulate the information included in the packet and forward the data without a packet format to the external device connected to it. Though, the external device must form a bluMiU data packet in order to send data to the server.

When the client sends an ACL packet to the server, the BT module, the same way as before, forwards the packet to the host. Then the server shall transcribe the ACL packet

---

<sup>1</sup> This feature is not supported by the used BT module

received to a bluMiU data packet (the packet's format is shown in Figure 4-1) and transmit it to the external device.



**Figure 4-3 BluMiU multiple client architecture**

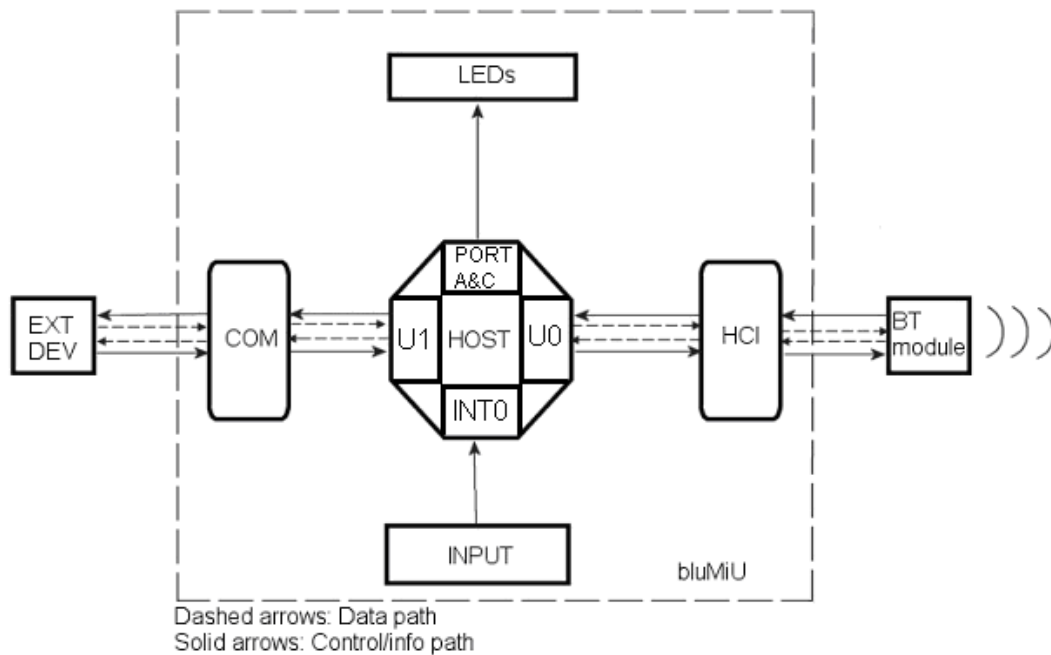
By using this protocol, many clients can be served without any handshaking or setup of data links. The server will be able to recognize the source of the data packet and send data to every of its clients. (Figure 4-3) It could also broadcast data packets to all of its slaves, if that feature, along with point to multipoint connections, were supported by the BT module. The hardware design that gives the ability to the system to support the bluMiU protocol and provide an external device connected to the bluMiU system with Bluetooth connectivity will be described in the next section.

## 4.2. *BluMiU hardware*

BluMiU hardware architecture follows the previous separation in terms of its external interfaces. (Figure 4-4) However, the new and improved features that it added to the existing system, required substantial changes in the hardware components of the system. The hardware components will be described in the next sections.

### 4.2.1. **HOST**

The HOST, as seen in chapter 3.2.1 (33) offers the processing power needed to execute the upper Bluetooth protocol layers and is the most important component of the system.



**Figure 4-4 bluMiU hardware architecture**

The tasks it must be able to successfully accomplish every time it is asked include

1. to communicate with the BT module via UART
2. to communicate with an external device via UART
3. to execute the commands given to it from a button input
4. to discover BlueTooth enabled devices in its vicinity, [client only]
5. to connect to them, [client only]
6. to disconnect from them,
7. to exchange data packets with any one of them,
8. to manage the data flow between the BT module and the external device connected to it (EXT DEV),
9. to correctly decode the incoming control and/or data packets, either they are being received from the BT module or the EXT DEV,
10. to act as a bridge, transforming the bluMiU specific data packets received from the EXT DEV to ACL data packets, compatible with the BlueTooth protocol, and vice versa and
11. to give commands to the BT module through the push-button input or through the push-button emulation from the EXT DEV and display their success or failed status on the LEDs

The characteristics that the bluMiU design has (mentioned above and in the introduction of this chapter) could not be supported by the previous microcontroller (mC) used (AT90S8515), so another one would have to be used. The selected mC was the ATmega161. An overview of the new AVR's characteristics and how it is used in the design are presented in Appendix A (page 89). The following reasons made the AVR selected a good choice:

- ✓ Availability
- ✓ Reliability
- ✓ Low cost
- ✓ Pin compatible with AT90S8515
- ✓ Fulfills the hardware requirements set by the bluMiU design, which are presented on Table 4-1.

**Table 4-1 Old vs new microcontroller**

<i>Microcontroller</i>	<i>Needed</i>	<i>AT80S8515</i>	<i>ATmega161</i>
<b>SRAM</b>	700bytes <sup>1</sup>	✗ 512bytes	✓ 1024bytes
<b>UARTs</b>	2	✗ 1	✓ 2
<b>Program memory</b>	12Kbytes	✗ 8Kbytes	✓ 16Kbytes
<b>Multiplier</b>	1	✗ 0	✓ 1
<b>I/O lines</b>	28	✓ 32	✓ 35
<b>Registers</b>	32	✓ 32	✓ 32
<b>External interrupts</b>	1	✓ 2	✓ 3
<b>16 bit timers</b>	1	✓ 1	✓ 2

### 4.2.2. BT module

The other essential component of the bluMiU system couldn't be other than the one that provides the system with BlueTooth connectivity, which is the BT module. The BT modules that were donated to the MHL by Ericsson Hellas were used both in bluMiU and in the BlueApple-BlueBridge. They are the hardware part of the Bluetooth

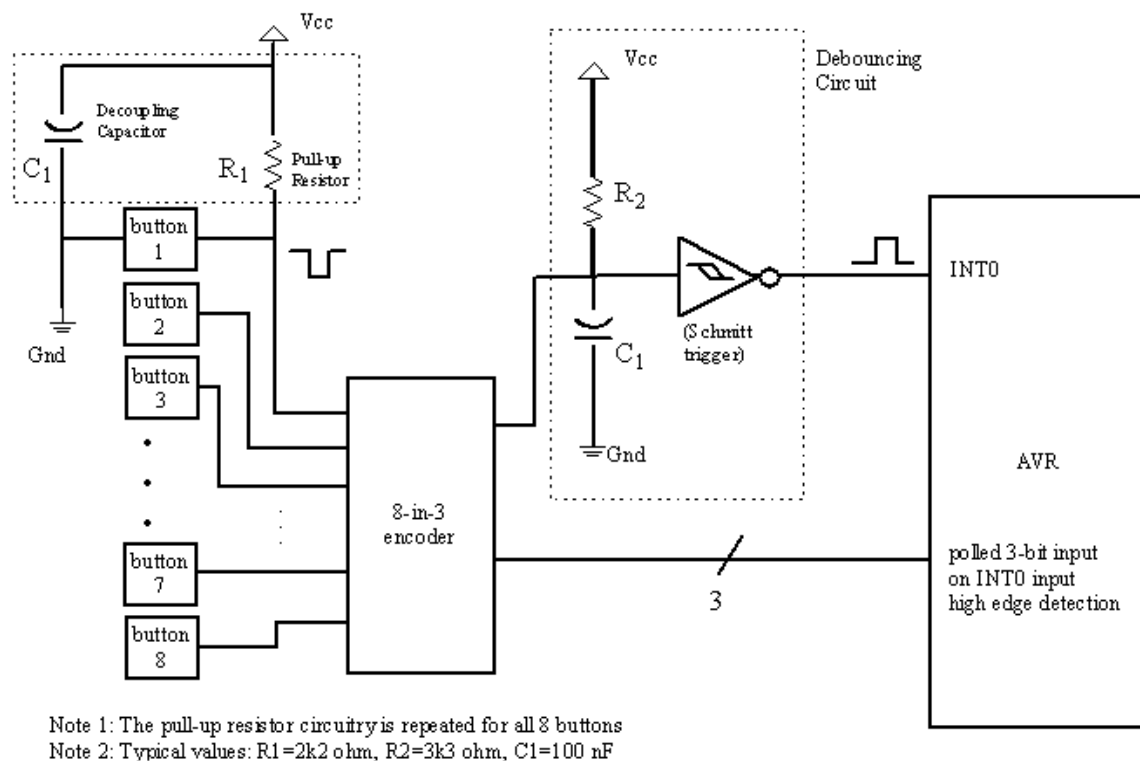
---

<sup>1</sup> At least 2x256bytes for UART buffers, 10x10bytes for inquiries, 7x8bytes for connections

Application Tool Kit, LZT 108 4123 R2A, developed by Teleca Comtec and are described in detail in Appendix B (page 92). As seen in the previous chapter, they embed the Bluetooth Radio, the Baseband, the Link Manager, a UART and a USB HCI. This means that the BT modules used accept Bluetooth commands, execute them and return their outcome. The UART interface is used as the Host Controller Interface.

### 4.2.3. HCI & COM

The mC makes use of the HCI and COM interfaces to communicate with the BT module and the EXT DEV. The internal UARTs of the ATmega161 are used to communicate with both of them. A default baud rate of 57.6Kbps is used and 256byte buffers are associated with the mC's UARTs, implemented in software.



**Figure 4-5 BluMiU input interface**

### 4.2.4. INPUT

The INPUT system used in bluMiU is almost the same as that of BlueApple-BlueBridge. It consists of 8 push-buttons. This was a choice made in the BlueApple design; it needed 5 of these 8 buttons and more buttons would supposedly help future work. The signals of these 8 buttons are driven to an 8-in-3 encoder so that fewer input lines of the



mC would be used (3 instead of 8). The encoder provides a signal that indicates that a button has been pushed. This is connected to an external interrupt line of the AVR, so, when it is triggered, the mC samples the 3 signals from the encoder and determines which button was that so that the appropriate action will be taken. (Figure 4-5)

### 4.2.5. LEDs

During the development of bluMiU, the improvement of the BlueApple-BlueBridge interface with the most important hardware components of the design (the BT module and the external device) and the speed-up of its data transfer rate to these were in the center of attention. A display that would simply show the success or failure of commands issued to the BT module should be enough. Having that in mind, the BlueApple-BlueBridge display and push button systems were used almost without any change in bluMiU. Thus, the display consists of two groups of 8 LEDs. The one group is used to count selections, in a binary scale, while the other is used for command and status messages.

### 4.2.6. EXT DEV

A device can be connected on the HOST, so that it can either acquire BlueTooth connectivity, or control the HOST so that it will issue BlueTooth commands to the BT module implemented in the HOST, or use the provided by bluMiU data transfer protocol so that it can exchange data with other devices that support the bluMiU data transfer protocol. This connection is achieved through the COM interface, which practically is the UART1 of the mC.

### 4.2.7. An estimation of the bluMiU hardware components' cost

The following hardware components were used for the development of each one of the bluMiU kits (client or server):

- 1x Atmel AVR ATmega161 microcontroller (\$7)
- 1x BlueTooth Application Tool Kit LZT 108 4123 R2A (\$60)
- 1x 74F148 8-Line to 3-Line Priority Encoder (\$0.25)
- 1x DM74LS14 Hex Inverter with Schmitt Trigger Inputs (\$0.25)
- 1x ADM202EAN RS-232 Line Drivers/Receivers (\$1.46)

- Various connectors, capacitors, push buttons and LEDs (\$1.5)
- PCB construction (\$20)

By using these components the cost of the platform's hardware would be at least \$90.

Since a low-cost Bluetooth design is a primary objective of this thesis, the cost could be further reduced by replacing the application tool kit by a Bluetooth RF IC and a Bluetooth Baseband controller. The Bluetooth RF and the Bluetooth baseband controller would have to be connected to each other though, so that a Bluetooth module would be constructed, that would offer the same services as the ready made one. The minimum cost for a Bluetooth Baseband controller was found to be \$0.87 for the Xemics SA XE1402 Bluetooth baseband Controller, while the minimum cost for a Bluetooth RF IC is \$0.5 for the Skyworks Solutions Inc.<sup>1</sup> SKY73001 Bluetooth RF Transceiver. The XE1402 is a Bluetooth baseband controller that supports only data transfers (not voice) and is BT spec 1.2 compliant, while the SKY73001 is a full featured Bluetooth RF IC. Both ICs are targeted for low cost, low power and small size designs. Using these, instead of a Bluetooth evaluation kit, would make the hardware components of the design cost more than \$50 less.

A protocol definition and some hardware components would never be enough for a complete system to be constructed. The software enables the protocol to run on the hardware components, making it a very important element of the system. The following sections specify the software architecture of the bluMiU system.

### **4.3. *BluMiU software***

The development of the software that was embedded in the bluMiU host was the most complicated and time consuming task throughout this thesis' development. Meeting the initial requirements that were set in the beginning of the development needed very sophisticated software architecture with very carefully written source code that would be able to deal with multiple events (interrupts) almost simultaneously and serve all of them

---

<sup>1</sup> Skyworks Solutions Inc. was created by Conexant Systems Inc. in a merger with Alpha Industries Inc as a wireless semiconductor company [<http://www.conexant.com>]

in real time. The software architecture of the client and the server is shown in Figure 4-6.

The idle process makes the mC to enter a power saving mode until an interrupt is triggered. When that happens, the mC will execute the appropriate software module. After each of the modules completes its jobs, the control is returned to the idle state, where it waits for other interrupts. In the following sections each software module will be thoroughly described and all its exact characteristics will be presented. The most important software modules that enable the bluMiU protocol to function are the UART control modules (UART0 and UART1). They will be specified in the following section.

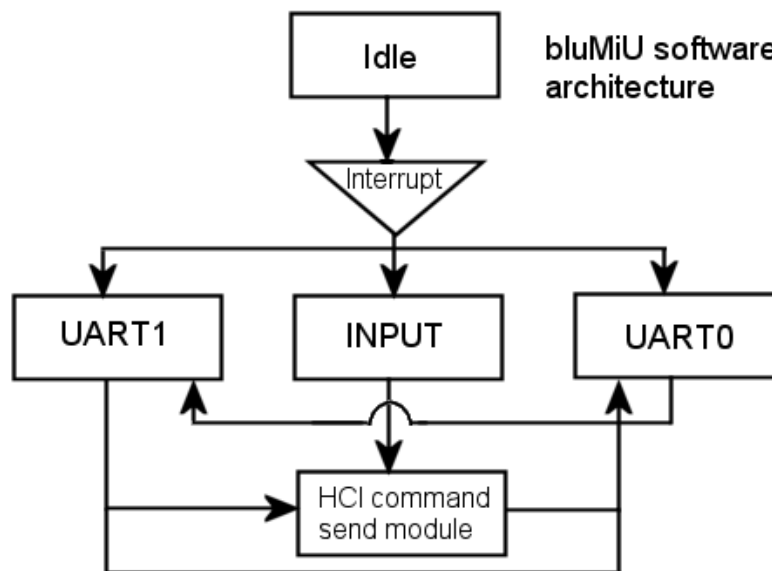
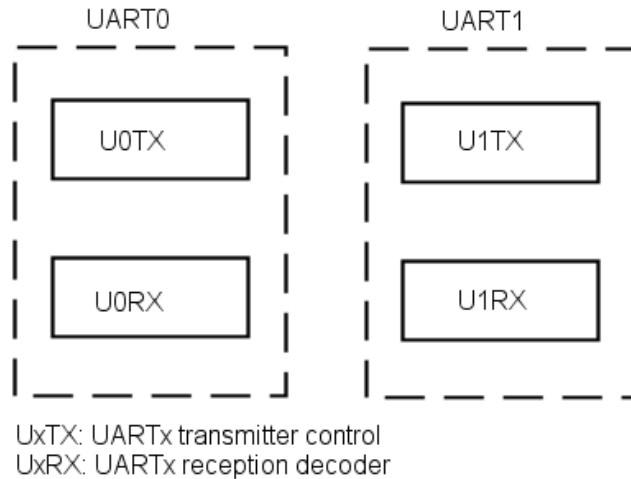


Figure 4-6 bluMiU software architecture

### 4.3.1. UART control modules

The UART control modules undertake the most important task in the bluMiU design; they must receive, decode and transmit packets to the BT module or the EXT DEV. Infallible function of these modules is necessary because the design's credibility relies almost solely to these. The system has 2 UARTs, one dedicated to the communication between the mC and the BT module and the other between the mC and an external device. Every UART can receive and transmit bytes. Four software modules, one module for each of the two receivers (UxRX) and two transmitters (UxTX), have been implemented. (Figure 4-7)



**Figure 4-7 UART software modules**

The two UART reception decoders will be described in the next sections. At first their common elements will be examined, and then their functionality will be specified.

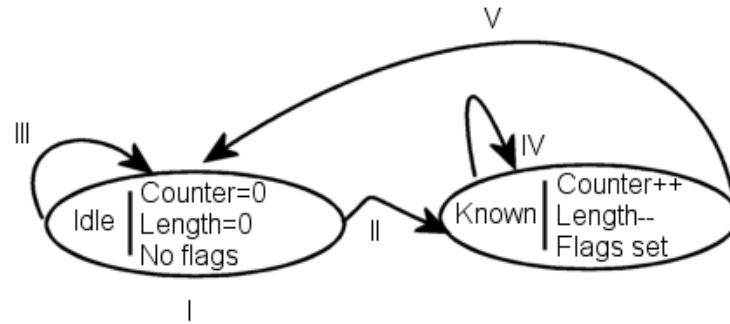
### **4.3.1.1. UART reception decoders**

The UART reception decoders (U0RX, U1RX) are the highest level software modules included in the bluMiU system. They run all the Bluetooth protocol layers that are embedded in the system, give Bluetooth functionality to the system and enable data exchange between remote bluMiU devices.

All the UART decoders implemented in the bluMiU design have the same binary tree with states structure. The decoders implemented are:

- The U0RX: It decodes packets being received from the BT module
- The U1RX: It decodes packets being received from the EXT DEV.

Their states are held until the whole packet has been received. Every received byte that belongs to a packet changes the state of the decoder. Registers are used to hold the decoding states. They count the number of bytes that have been received (COUNTER), the number of bytes remaining to complete the packet (LENGTH) and hold flags. A generalized state diagram of the decoders is depicted in Figure 4-8.



**Figure 4-8 UART decoder state diagram**

- I. The decoder starts in an idle state, where no flags are set, LENGTH and COUNTER are zero. When a byte is received it is checked whether it is the packet indicator of a known packet.
- II. If the byte is a known packet indicator, a special flag is set and the COUNTER is increased so that the decoder will decode the next byte correctly, since it will know that it will be the 2<sup>nd</sup> byte of this specific packet type.
- III. If the byte cannot be recognized as a valid indicator, nothing happens and the decoder returns to the idle state.
- IV. As bytes arrive, the COUNTER is further increased and the LENGTH is decreased. This phase of the decoding is the most important one, since it is the one where received data is analyzed and appropriate actions are taken (e.g. data essential for the connection creation is written in the SRAM or packets are transcribed between the BlueTooth and bluMiU formats). These actions depend on the byte received and the flags set.
- V. When LENGTH reaches 0 the packet reception has completed, so the COUNTER is set to 0 too, all the flags are cleared and the decoder returns to the idle state. Bytes received after the decoder finishes decoding of a whole packet and has returned to the idle state, will be considered as the packet indicator of a new packet and will be checked appropriately.

Of course, the structure of the decoders is far more complicated than that depicted in Figure 4-8. In the following sections we will see the tasks that each of the UART decoders undertake.

### 4.3.1.2. UART receiver (from the BT module) [U0RX]

One of the most important tasks that bluMiU undertakes is the reception of packets from the BT module and its capability to decode the bytes forming them, since Bluetooth connections can only be established by correctly decoding the appropriate Bluetooth packets.

The purpose of the U0RX is twofold:

1. It decodes the incoming events that the BT module transmits to the Host
2. It transcribes the ACL data packets, which are targeted for the EXT DEV, to the bluMiU specific format and retransmits them (through the other UART) to the EXT DEV

#### Event decoding

By decoding the events, U0RX collects and stores to the RAM the information about the devices in the BT module's vicinity and the connected to it devices. It also notifies the EXT DEV for the disconnections, as they happen. Furthermore, U0RX uses the display system to output the success or not of a command issued to the BT module.

All the events supported in bluMiU are implemented according to the BT spec 1.2.

The U0RX decoder supports the following events:

Event	Event code (hex)	Length (bytes)	Parameters
Inquiry complete <sup>1</sup>	0x01	4	Status
Inquiry result <sup>1</sup>	0x02	18	# of Responses, BD ADDR, Page Scan Repetition Mode, Page Scan Period Mode, Reserved, Class of Device,

---

<sup>1</sup> Only the client supports this event

## Microprocessor & Hardware Laboratory

			Clock Offset
Remote name request complete <sup>1</sup>	0x07	258	Status, BD ADDR, Remote Name
Connection complete	0x03	14	Status, Connection Handle, BD ADDR, Link Type, Encryption Mode
Disconnection complete	0x05	7	Status, Connection Handle, Reason
Command complete	0x0e	6+n	# of HCI Command Packets, Command Opcode, n bytes of Return Parameters
Command status	0x0f	7	Status, # of HCI Command Packets, Command Opcode
# of Completed Packets	0x13	8	# of Handles, Connection Handle, HC # Of Completed Packets

For each event decoded, the U0RX takes different actions. Usually, when an event is decoded, the U0RX indicates, on the LEDs of the display system, the event's status (where applicable) and the event that has happened. The one group of 8 LEDs that are driven by the mC are used. The upper 4 LEDs indicate the event's identity and the lower 4 show the event's status or other significant event parameters.

The following output actions are taken for every event supported:

Event	Output (Upper 4)	Output (Lower 4)
Inquiry complete	2 <sup>nd</sup> LED	# of devices
Inquiry result	2 <sup>nd</sup> LED	# of devices
Connection complete	3 <sup>rd</sup> LED	2 <sup>nd</sup> LED
Disconnection complete	3 <sup>rd</sup> LED	1 <sup>st</sup> & 2 <sup>nd</sup> LED
Remote name request complete	3 <sup>rd</sup> LED	1 <sup>st</sup> LED
Command complete	1 <sup>st</sup> LED	Status byte
Command status	3 <sup>rd</sup> LED	Status byte
# of Completed Packets	Nothing	Nothing

Connection related information retrieved from the events supported is stored in the memory (or removed from the memory in the case of a disconnection). The stored in the memory connection related information is accessible to the EXT DEV.

### **Data packets**

The U0RX decoder is the software module that undertakes the task to transcribe the Bluetooth data (ACL) packets received from the BT module and to retransmit them to the EXT DEV.

In the client the data included in the ACL packets is de-capsulated from the ACL packet and is transmitted to the EXT DEV. The host only checks if the identity of the ACL packet is the expected one. If it isn't, the incoming ACL packet is ignored.

In the server the ACL packet is transcribed to a bluMiU packet and transmitted this way to the EXT DEV. This way, the client that sends the data will be known to the EXT DEV, as well as the amount of data the client transmitted to the server. The identity is again checked for validity; if it is not the expected, the ACL packet is ignored.

#### **4.3.1.3. UART receiver (from the EXT DEV) [U1RX]**

Another very important software module of bluMiU is the decoder of the packets coming from the EXT DEV (U1RX). It can receive the following types of bluMiU packets:

- Data packets to transmit them wirelessly to a bluMiU device
- Manual Bluetooth commands from the EXT DEV to the BT module
- Packets indicating an Bluetooth command implemented in the mC to be issued to the BT module (button emulation)

The communication is carried out in 57.6Kbps by default, as also happens with the BT module.

### **Data packets**

Data packets that are transmitted to the bluMiU by the EXT DEV must have the format described in section 4.1.2 (page 46). After the U1RX detects that a data packet is being received, it will transcribe the packet to a Bluetooth protocol compatible ACL data packet (by adding to it the ACL header and removing the bluMiU packet indicator) and



transmit it through the UART0 to the BT module, so that it will wirelessly transmit it to the target bluMiU device (if it exists). The validity of the handle that has been received, which indicates the target bluMiU device, will be checked by the BT module. The received packet's header is just checked for its compliance with the format of the header specified by bluMiU.

### Manual Bluetooth commands

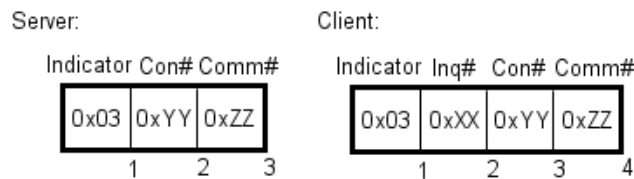
BluMiU also accepts packets including Bluetooth commands, so that it can retransmit them to the BT module. These packets have the format of Figure 4-9. After the U1RX detects the Manual Bluetooth packet, it forwards the command contained in the packet to the BT module without checking the command itself. By using this service, the EXT DEV can issue to the BT module virtually every command specified in the Bluetooth protocol, allowing the EXT DEV to gain almost full control of the BT module.



**Figure 4-9 Manual Bluetooth command format**

### Button emulation

The EXT DEV can use the services, normally provided by bluMiU by the use of push-buttons, by bypassing the button system -as well as the need for a human hand- and sending to bluMiU a 3 or 4 byte packet for the server and client respectively. This packet's format is depicted in Figure 4-10. It consists of a packet indicator and 2 or 3 more bytes, the latter for the client and the first for the server bluMiU module.



**Figure 4-10 Button emulation packet format**

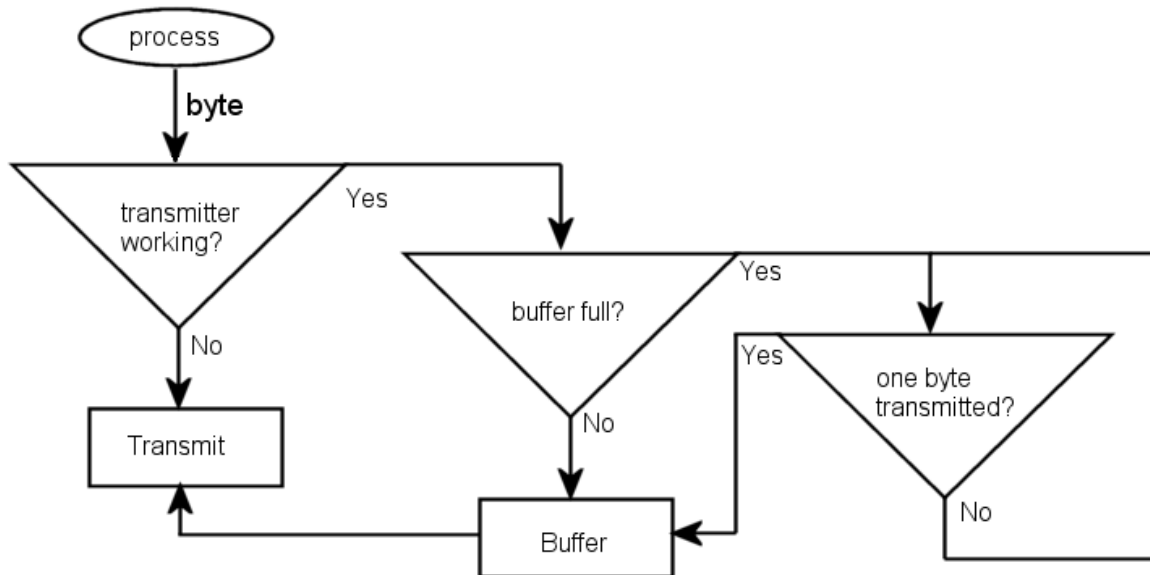
These bytes have the following function:

- The inq# and conn# fields select a remote device that has been found in the vicinity (or is connected) to the system.
- The comm# field selects a command implemented in bluMiU to be executed by it.

The previously described packet format (push-button emulation) has been designed to act as an addition to the existing push-button control software so that it would help bluMiU to be able to function without the explicit need for a human user. The push button emulation uses the same mechanism as the control for the actual buttons.

### 4.3.1.4. UART transmitter control

The BT module communicates with the mC through the UART. The EXT DEV also uses this way to exchange data with the mC and/or control it. Since the beginning of bluMiU development it was realized that without a good “driver” for the mC’s UARTs, improvement in the BlueAppleE-BlueBridge performance would never be achieved and new services could not be added, making the UART mechanism the most important software module of bluMiU. A buffering mechanism in both of the UARTs transmitters was implemented. Two 256byte buffers are allocated in the mC’s RAM, one for every UART.



**Figure 4-11 BluMiU buffering mechanism (UxTX)**

By using the buffering mechanism depicted in Figure 4-11, when an application requests to transmit a byte from a UART, the mechanism first checks to see if this UART is in the process of transmitting another byte.

- If the UART is not transmitting, the byte is given to the transmitter, so that it will be transmitted at once. It takes some time for the transmitter to transmit a byte, so it is

very probable that another request to transmit a byte will be made while the transmitter is working.

- If the UART is currently in the process of transmitting another byte, the new byte cannot be transmitted immediately and shall be handled according to the buffer's state.
  - If the buffer is not full, the new byte is stored in the buffer so that the UART will transmit it when it has finished transmitting all the bytes that are stored in the buffer before that byte.
  - If the buffer is full, the UART mechanism will be stalled until a byte from the buffer has been transmitted. Until then, the new byte is stored in the mC's memory along with other data essential to the UART mechanism. When a byte has been transmitted, the new byte will be written in the buffer and the mechanism will resume its normal function.

At the time the UART finishes transmitting a byte, a UART transmit complete interrupt is triggered (TX). In the TX interrupt service routine (ISR), the UART mechanism checks if the buffer is empty.

- If it is not empty, the oldest byte stored in the buffer is transmitted.
- If the buffer is empty, the UART mechanism stops the transmitter. The next byte that will arrive to the UART will be transmitted at once.

We saw that buffering a buffering mechanism has been implemented only for the transmitters of bluMiU. Buffering did not need to be implemented in the receivers because the mC is fast enough so that it can process a received byte completely before the next byte arrives in the UART receiver, even if the system is manipulating byte streams belonging to two different packets at the same time (one from the EXT DEV and one from the BT module). The maximum CPU time for a decoder to fully decode a byte that has just been received is 23.6usec<sup>1</sup> (the last byte of the header of an incoming bluMiU data packet from the EXT DEV, which needs to be transcribed to a BlueTooth

---

<sup>1</sup> Most of the commands implemented in the AVR need a single clock cycle to execute [14] (even for SRAM transfers) and an 11.0592MHz clock is used, so the AVR's throughput is approximately one command every 9nsec

ACL data packet, so that it will be transmitted by the BT module to a connected device). Multiple interrupts can be processed by the mC, while the decoders process a byte. Though, all the ISRs implemented in bluMiU complete in a maximum time of 23.6usec and a nominal time of 4usec. Two consecutive bytes are received from the mC's UART in 0.13msec (with a data rate of 57600Kbps), making the existence of a buffering mechanism in the receiver meaningless, since, even with a maximum system load, the decoders are faster than the UART. The mC, either way, can wait with one byte in its data reception register for some usec, until the decoders end their jobs.

The UART0 and UART1 control mechanisms have been described. They cover the largest part of the bluMiU source code. In the next sections the rest of the software modules are described. They are the command send module and the input control module.

### **4.3.2. Command sending (to the BT module)**

A BlueTooth device must be able to connect to other BlueTooth devices and use their services, by using the means provided by the BTspec. In the previous chapters we saw that BlueTooth commands can be issued to the BT module through the bluMiU system from an external device. Though, some commands crucial for locating, connecting and identifying remote devices have been included in the software so that control from an external device would not be necessary.

The implemented in bluMiU BlueTooth commands are selected through indexing. The commands' byte sequences have been included in the bluMiU program code, which is stored in the mC's non-volatile flash memory, leaving the SRAM available for the UART buffering and the connection data.

A very large number of BlueTooth commands has been specified in the BT spec. Though, the microcontroller used has memory limitations that wouldn't allow many of them to be implemented. In order to reduce the memory used for storing them, only the commands that are essential for the tasks of discovering BlueTooth devices, connecting to them and exchanging data with them were included in the bluMiU design. Commands are implemented according to BT spec 1.2. The supported commands in the server are:

## Microprocessor & Hardware Laboratory

HCI command	Details
Disconnect	Terminates an existing ACL connection based on the connected device's Connection Handle
Reset	Resets the Host Controller (HC) & the LM. The local Bluetooth device enters stand-by mode
Set Event Filter	Specifies different event filters (the Host receives only events that interest it)
Change Local Name	Gives a user-friendly name to the local Bluetooth device
Read Local Name	Reads the user-friendly name of the local Bluetooth device
Write Scan Enable	Writes the parameter deciding whether a device will perform periodic inquiry and/or page scans so as to be visible by remote devices, or not
Read Buffer Size	Reads the max size of the ACL & SCO packet payload that the HC can receive from the host
Read BD ADDR	Reads the BD ADDR of the local device
Ericsson Set UART BR	Changes the baud rate of the UART (HCI transport). Default value: 57.6Kbps

The supported commands in the client are:

HCI command	Details
Inquiry	Searches for active Bluetooth devices in the vicinity and returns relative information
Create Connection	Attempts to establish an ACL connection to another device based on the remote device's BD ADDR (other information found by an inquiry command are helpful). On success, a Connection Handle is assigned to the ACL link
Disconnect	Terminates an existing ACL connection based on the connected device's Connection Handle
Remote Name Request	Obtains the user-friendly name of a remote device based on its BD_ADDR
Reset	Resets the HC & the LM. The local Bluetooth device enters stand-by mode
Set Event Filter	Specifies different event filters (the Host receives only events that interest it)
Change Local Name	Gives a user-friendly name to the local Bluetooth device
Read Local Name	Reads the user-friendly name of the local Bluetooth device
Read Buffer Size	Reads the max size of the ACL & SCO packet payload that the HC can receive from the HOST
Read BD ADDR	Reads the BD ADDR of the local device
Ericsson Set UART BR	Changes the baud rate of the UART (HCI transport). Default value: 57.6Kbps

After the issue of every one of the commands implemented into the HOST<sup>1</sup>, the BT module returns to it an event appropriate for the specific command. All the events that can be returned by the supported by bluMiU Bluetooth commands have been implemented in bluMiU. They are described in a previous section of this chapter (4.3.1.2 on page 57).

A means for issuing the implemented Bluetooth commands to the BT module has been seen in section 4.3.1.3 (page 59), as the button emulation. The actual buttons (which control the bluMiU and command it to issue commands to the BT module) are controlled by the input control software that will be described below.

### 4.3.3. INPUT control software

The push buttons are the most common user interface of an embedded system. In the system developed in this thesis, the existent input system has been retained, as it was seen in section 4.2.4 (page 51). When a button has been pushed an external interrupt of the mC is triggered. Then, the ISR of this interrupt is executed (INT0) and it finds which of the 8 buttons has been pushed.

In bluMiU, only 4 of the 8 buttons are used. The 4 buttons that are not used have been retained in the bluMiU design, but removing them is a trivial task. The function of each of the 4 buttons is:

**Button 1:** Selects a Bluetooth command, an inquired device, or a connected device depending on the mode, which is selected by *Button 3*. *Button 1* increases by one the selection. The selection is shown on the port A LEDs.

**Button 2:** Same as *Button 1*, except that it decreases by one the selection.

**Button 3:** Selects an operation mode for the rest of the Buttons.

---

<sup>1</sup> The only exception is the Ericsson Set UART BR command; this isn't a BT spec specified HCI command.

## Microprocessor & Hardware Laboratory

- For the client there are 3 different modes:
  - *Mode 0*: HCI command operation mode. *Buttons 1* and *2* select the HCI command, while *Button 4* transmits it to the BT module.
  - *Mode 1*: Inquired device operation mode. *Buttons 1* and *2* select an inquired device, while *Button 4* transmits its BD ADDR to the EXT DEV.
  - *Mode 2*: Connected device operation mode. *Buttons 1* and *2* select a connected device, while *Button 4* transmits its connection handle and its BD ADDR to the EXT DEV.
- For the server there are 2 different modes:
  - *Mode 0*: HCI command operation mode. *Buttons 1* and *2* select the HCI command, while *Button 4* transmits it to the BT module.
  - *Mode 1*: Connected device operation mode. *Buttons 1* and *2* select a connected device, while *Button 4* transmits its connection handle and its BD ADDR to the EXT DEV.

**Button 4:** Action button. The action that is described by *Button 3* is taken.

Having examined the hardware and software parts of bluMiU in detail, the testing procedures used to validate the bluMiU hardware and software design can be seen.

## 5. Testing & Validation

---

When the construction of a system has been completed, it is imperative that a well defined set of tasks is used to validate the system and demonstrate that the system works. Also, during the construction of a complex system, such as bluMiU, after the completion of every subsystem's design and implementation, tests are also necessary because:

- The majority of its bugs must be found *before* including it in the system
- Its correct cooperation with other subsystems of the system must be thoroughly tested
- Verification of the whole design must take place after the subsystem's implementation is completed, if that is possible.

The system constructed in the present thesis is formed by a number of hardware and software subsystems, which were specified in detail in the previous chapter. Their test and validation procedures during the implementation of bluMiU, as well as the validation of the whole system will be presented in this chapter.

Testing procedures that took place before the total design was constructed were the most important tests of the system. Making extended tests to each subsystem during the implementation provided the designer of the bluMiU system the ability to use each subsystem as a component, without any doubts if its function was correct or not.

### **5.1. BT module validation**

Most of the components forming the hardware around the HOST (the BT module, the INPUT system and the HCI interface) were inherited from the BlueAppIE system and, thus, did not need extensive tests, except a simple verification that they do function as expected.

Though, the BT module and its HCI interface were fundamental for the correct operation of the system. In order to reassure their correct operation and make sure that the



requirements of bluMiU by them would be fulfilled, they were exhaustively tested before starting the implementation of the system. The tests for the BT module were necessary because it was the only component that could provide Bluetooth connectivity and doubts whether its operation was infallible had emerged during the BlueApple development. Another issue that was also tested was if it could offer the services introduced by bluMiU, during the development of the latter.

The BT module, as well as its UART interface to the HOST (HCI), was put under trial in the context of the following test procedures:

- By connecting a computer and the BT module through the UART (HCI) interface, simple HCI commands (these were the Reset, Read BD ADDR, Read Local Name, Write Scan Enable, Read Scan Enable, Read Local Supported Features and Read Buffer Size HCI commands) were transmitted to it from the computer, by a terminal application, and the module's replies (HCI events) were checked for validity, according to the BT spec 1.2.
- Using the same computer-BT module connection as before, the module's ability to locate other Bluetooth devices and connect to them was tested, by using the following HCI commands:
  1. Set Event Filter
  2. Write Scan Enable
  3. Inquiry
  4. Create Connection
  5. Disconnect

These commands were used to establish a connection with a Bluetooth enabled mobile phone (Ericsson T39). The module's ability to request a Bluetooth connection from the mobile phone and accept a connection request made by the mobile phone, were validated.

- The speed of 460kbps data rate that the BT module specifications claim to support was validated using its USB interface. The BT module was connected via a USB cable with a computer. Then the module exchanged large files with another USB Bluetooth module provided by CSR. Drivers provided with the Ericsson Application & Training Toolkit were used for the BT module.

These preliminary tests on the BT module certified that it could be used in the bluMiU design.

After the implementation of the first versions of the software of the system that could transmit data through the UART, the BT module-HOST communication was tested. This test was accomplished by connecting a serial cable's RX and GND signals to the RXD0 and GND line of the HOST. The other end of the serial cable was connected to a computer running a terminal application that could display the received by the UART bytes in a hexadecimal format. This way, the events that the module would transmit to the HOST in reply to the HCI commands it sent, were checked for correctness and compliance with BT spec 1.2. (Figure 5-1)

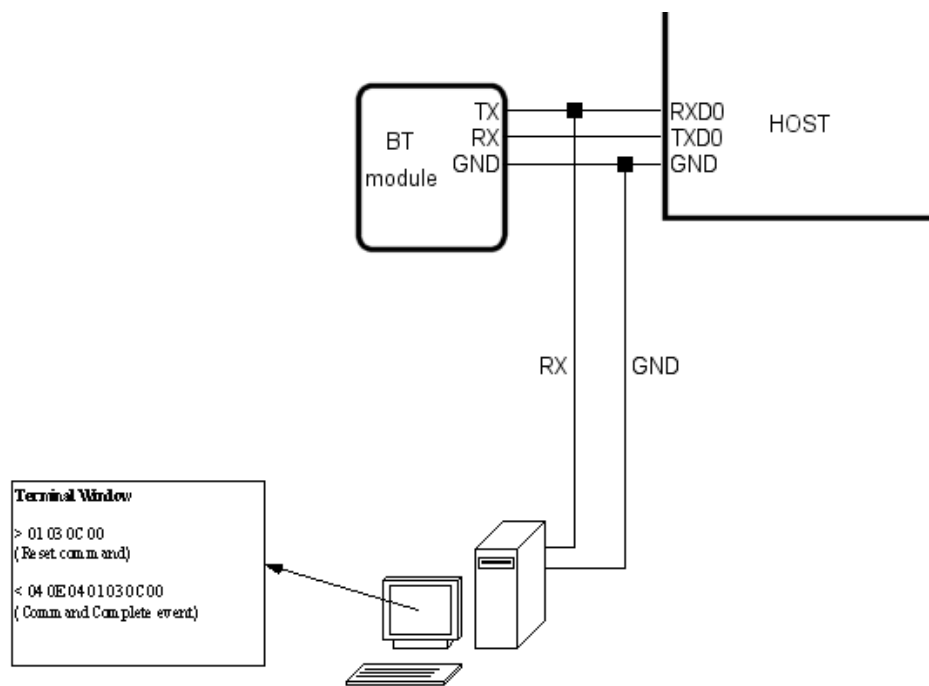


Figure 5-1 BT module-HOST communication monitoring

Every one of the subsystems had to be tested exhaustively so that the correct cooperation with the rest and the final flawless function of bluMiU would be guaranteed.

### 5.2. Software validation

The software design of the bluMiU system is modular. That way, every module could be tested independently from the others before being used in the system. The test and validation procedures followed a hierarchical manner. The simpler and more

fundamental software modules were tested first. Thus, the first modules under trial were the UART transmitters, because every other module uses that.

### **5.2.1. UART transmitter validation**

As it was stated in chapter 4.3.1.4 (page 61) the efficient transmission of data was a primary requirement for the bluMiU. The buffering mechanism that was implemented underwent many changes and needed constant debugging throughout this thesis' implementation.

In order to debug the UART mechanism a UART connection between the HOST and a computer was used. The HOST transmitted HCI commands to the computer through a serial cable. The data arriving to the computer (in a terminal application, same as before) should be the same as that supposedly transmitted from the HOST.

Later in the development, tests for the correct transmission of data from the HOST to the BT module, through the UART mechanism were needed. To accomplish that, a serial cable's RX and GND signals were connected to the TXD0 and GND line of the HOST, in the same way as in the BT module monitoring that was described above and depicted in Figure 5-1, except that the TXD0 line of the HOST is connected with the computer, not the RXD0. In this way, the packets sent by the HOST to the BT module could be monitored.

UART1 (that transmits data to the EXT DEV) has an identical mechanism with UART0, so monitoring on it was not explicitly needed. After the UART transmitters' operation was verified, the next most frequently module used was the input decoder module.

### **5.2.2. INPUT decoder validation**

Since the beginning of the bluMiU design, the INPUT subsystem should work infallibly without even a minor bug, because this was the only subsystem that could control the system in the early stages of bluMiU implementation.

The test of the decoder of the INPUT subsystem was accomplished, by the use of the LED system, the UART connection to the EXT DEV and the monitoring of the UART connection to the BT module.

- The primary test of the hardware part of the INPUT subsystem was merely the lighting of specific LEDs when the appropriate button was pushed
- The selection function of the buttons was tested by transmitting the selection's data through the EXT DEV UART connection
- The command send function of the buttons was tested by validating that the correct HCI command was transmitted to the BT module through the UART monitoring

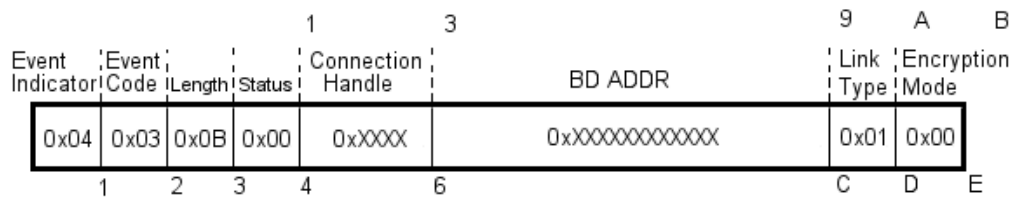
After the verification of the input decoder the UART reception decoders should be verified, so that the validation and testing procedures of the system's components would be completed.

### 5.2.3. UART decoder validation

The incoming packets from the BT module and the EXT DEV are decoded by U0RX and U1RX decoders respectively. The tests to validate the correct function of these decoders used the LEDs and the UART connection to the EXT DEV to report the results of the decoding procedures, along with the monitoring of the incoming packets to the decoders.

This test procedure can be better explained through an example: The BT module's replies are monitored by the system of Figure 5-1 and a Connection Complete event is being transmitted to the HOST, either after accepting a Connection Request from a remote BlueTooth device or after a remote BlueTooth device accepted a Connection Request by the local HOST. The Connection Complete event indicates to both of the Hosts forming the connection that a new connection has been established. This event also indicates to the Host, which issued the Create Connection, or Accept Connection Request or Reject Connection Request command and then received a Command Status event, if the issued command failed or was successful. [3] The Connection Complete event format that is expected by bluMiU (according to BT spec 1.2) is depicted in Figure 5-2. After verifying that the Connection Complete event has been transmitted by the BT module to the HOST, the correct decoding by the U0RX decoder

in the HOST should be verified. This is accomplished by lighting various LEDs indicating that fields received had the expected values and by transmitting to the computer connected to the EXT DEV bluMiU port the Connection Handle and the BD ADDR of the device connected to the bluMiU.



**Figure 5-2 Connection Complete Event**

The same test procedure was used for the U1RX and the U0RX decoders implemented in bluMiU. Of course, the ultimate test for the transmitters and the decoders would be the successful data exchange between two connected bluMiU devices. This would also be the validation of the whole bluMiU protocol and system that was designed. The test methods that were used to validate the bluMiU protocol were based on the test procedures mentioned in the previous sections and will be presented in the next section.

### 5.3. *BluMiU validation*

Through validating the subsystems composing bluMiU, the system itself was tested. The validation procedure of the integral bluMiU system and protocol was far more complicated than the validation of each of the subsystems (though, the same mechanisms as in the subsystems' tests were used) and consisted of testing all the functions supported by bluMiU. This final test procedure emulated a real-life application of the bluMiU system and protocol. More specifically the following capabilities of the system were validated:

1. The pre-connection setup of the bluMiU client and server
2. The discovery of various Bluetooth devices by the client and the identification of the server using the reports to the EXT DEV
3. The establishment of a Bluetooth connection between the client and the server
4. The exchange of data between bluMiU devices using the bluMiU protocol. Here an evaluation of its efficiency and comments on possible improvements on the protocol itself were made

### 5. The efficient annulment of BlueTooth connections involving bluMiU devices

The test procedure of each of the mentioned tests will be presented in the following sections.

#### 5.3.1. BluMiU pre-connection setup

Before establishing a connection between a client and the server, certain actions should be taken for both of the sides involving in the connection.

The power-up of the bluMiU server and client platform should be followed by no other events than the lighting of the HOST's idle LED and the one indicating that the first implemented command will be executed when commanded to. When these LEDs are lit, the HOST will be ready to receive commands from the INPUT or the EXT DEV (Manual HCI and Button emulation). The BT module is powered up along with the HOST, since it is powered by the bluMiU platform, and according to its datasheet there is no need for a power up sequence. [16] After the power up of the bluMiU platform, configuring of the BT module should take place.

- A user friendly name can be given to the modules by executing the appropriate command implemented in the HOST (the names are: MiU-000 to the server and MiU-001 to the client)
- The server should be visible in an inquiry scan. This is done by executing the Write Scan Enable command implemented in the server
- The server should set the auto-accept connection parameter of the BT module for every device, so that it would be able to connect with a client or another BlueTooth device automatically. This is accomplished by executing the Set Event Filter command implemented in the server

The commands used above should be able to be replaced (another name can be given, the client too can be scanned in an inquiry and different event filters can be set for both the client and the server<sup>1</sup>) and/or other (not implemented in the HOST) HCI commands

---

<sup>1</sup> For information on the functions of the Write Scan Enable and Set Event Filter HCI commands, refer to the BT spec 1.2. [3]

can be issued to the BT module by using the Manual HCI command mode from the EXT DEV. Finally, the HOST shouldn't get confused by the event of Number of Completed Packets that the BT module sends to the HOST every time the BT module receives successfully 5 packets.

The design in the pre-connection setup would be validated if the bluMiU server could be scanned in an inquiry scan, its name could be seen correctly and if it could accept the connection request from a BlueTooth enabled device. This was accomplished using some BlueTooth enabled mobile phones (Nokia 6230 and Ericsson T39).

Validation of the Manual HCI command and Button emulation modes was achieved by enabling the client to be scanned in an inquiry (the Write Scan Enable command is not implemented in the client), by executing the commands through the Button emulation and by executing the Read Supported Features and Read Buffer Size commands to the server. These commands' successful execution was validated through the UART monitoring of the Command Complete events that were returned for each one of them.

### **5.3.2. Discovery of BlueTooth devices by the client**

After correctly starting-up the bluMiU system, the client should be able to search its vicinity for BlueTooth devices, identify and connect to them.

By issuing the Inquiry command to the BT module, the BT module will report the BlueTooth devices it found to the HOST with an Inquiry Result event for every device<sup>1</sup> it found. The client HOST should decode every Inquiry Result event correctly and store the required for establishing a data connection fields to its SRAM. After at least one Inquiry Result has been received, each discovered device's characteristics can be requested through the Buttons or the Button emulation, after having selected the device. The HOST should be then able to report that inquired device's BD ADDR to the EXT DEV through the UART. The user friendly name of an inquired device would be

---

<sup>1</sup> BT spec 1.2 specifies that multiple devices can be reported in a single Inquiry Result event, but in the specific BT module only one device per Inquiry Result is reported.

requested by issuing a Remote Name Request to that device. The Remote Name Request Complete event should return the target inquired device's name and the HOST should transmit the name to the EXT DEV.

The bluMiU client's ability to locate BlueTooth devices and identify them was validated by finding the server and reporting its characteristics through the EXT DEV. The client's ability to support many inquired devices and report the selected device characteristics was validated by locating the server *and* the mobile phones that were mentioned previously and retrieving their BD ADDR, as well as reporting their user friendly names when they are requested for each device.

### **5.3.3. Establishment of BlueTooth connections**

After the client locates the BlueTooth devices in its vicinity and identifies them it should be able to connect with any one of them. The same should happen for the server, though the server can't be able to request a connection from a BlueTooth device, only accept connection requests.

By issuing the Create Connection command to the BT module the client should be able to establish a BlueTooth connection with any inquired device selected (by the Buttons or the Button emulation). After the connection has been completed, the BT module will return a Connection Complete event (Figure 5-2). The HOST should be able to decode that event correctly, store the connected device's connection handle and BD ADDR to the SRAM and report the selected connected device's characteristics to the EXT DEV. The server should be able to decode the Connection Complete events and report the connected device's characteristics to the EXT DEV as well.

This time, correct function of the bluMiU was validated by establishing a connection between the client and the server, as well as the client and the server with other BlueTooth devices, and reporting the correct characteristics of the device to the EXT DEV. Point-to-multipoint connections could not be achieved, due to the lack of their support from the BT module. [16]



### 5.3.4. Data exchange

Since the client and the server could be connected at will, the following data exchange matters could be tested and evaluated:

- The ability of the bluMiU platforms to actually exchange data
- The efficiency of the protocol used
- The virtue of any possible improvements in the bluMiU protocol, so that higher data rates would be achieved

#### 5.3.4.1. The ability to exchange data

In the requirements set for the bluMiU, the system should be able to accept data from an external device and transmit it wirelessly to another external device connected to it.

The bluMiU client or server should have the ability, after they connect to each other, to exchange data that is received from the external device in a full duplex manner. The data must be encapsulated in special packets with variable payloads. The way data transfers are done is described in detail in chapter 4.1.2 (page 46).

During the test processes in this stage, packets were formed and transmitted to the bluMiU server or client by the EXT DEV. The payloads of the packets ranged from 1 to 796 bytes, which is the maximum payload that an ACL packet can support. Validation of the bluMiU in this point meant that the bluMiU data packets would be transcribed correctly to ACL data packets, then they would be delivered correctly to the target bluMiU device (if the device was not valid they would not be delivered at all) and the decoding of data packets in the other end should transcribe the received ACL packets to bluMiU ones and transmit them correctly to the EXT DEV (in the client data is transmitted to the EXT DEV without a packet format).

The same methods of testing and validating all the subsystems that were seen in the previous sections were used here as well. The LEDs and UART monitoring were used to validate that the bluMiU packets are transcribed correctly to ACL ones and that the other end received the (correct) packets. In the place of EXT DEV were computers which were running a terminal application and through that the transcription of ACL packets to bluMiU packets was validated.

In the screenshot below (Figure 5-3) 12 packets that were transmitted by the bluMiU server are shown, after they had been received by the client. The packets transmitted were written in a file after the handle of the client was found. They had variable payloads ranging from 1 to 796 bytes (1, 2, 4, 8, 16, 32, 64, 128, 255, 256, 512 and 796). The packets were transmitted through COM 4 of the computer to the server bluMiU device and received by COM 1 of the same computer, using two instances of a terminal application that monitor the ports used. The packets were checked if the input from the file was the same with the output. By transmitting and receiving these packets correctly, the bluMiU server's data packet transmission mechanism and the bluMiU client's data packet reception mechanism were validated.

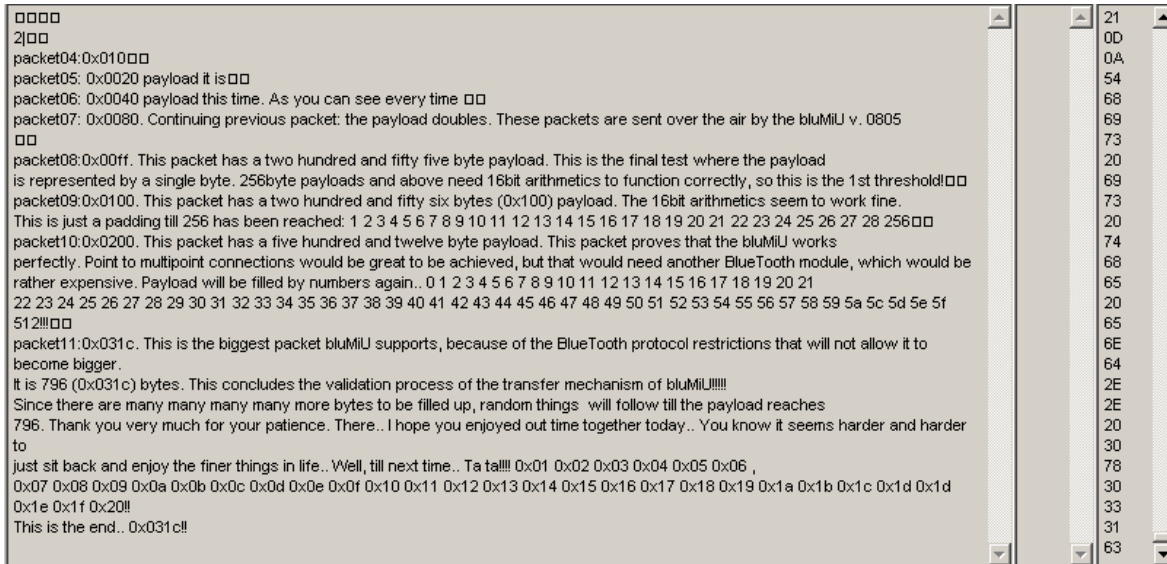
```

01
2packet03
packet04:0x010
packet05: 0x0020 payload it is
packet06: 0x0040 payload this time. As you can see every time
packet07: 0x0080. Continuing previous packet: the payload doubles. These packets are sent over the air by the bluMiU v. 0805

packet08:0x00ff. This packet has a two hundred and fifty five byte payload. This is the final test where the payload
is represented by a single byte. 256byte payloads and above need 16bit arithmetics to function correctly, so this is the 1st threshold!
packet09:0x0100. This packet has a two hundred and fifty six bytes (0x100) payload. The 16bit arithmetics seem to work fine.
This is just a padding till 256 has been reached: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 256
packet10:0x0200. This packet has a five hundred and twelve byte payload. This packet proves that the bluMiU works
perfectly. Point to multipoint connections would be great to be achieved, but that would need another Bluetooth module, which would be
rather expensive. Payload will be filled by numbers again.. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 512!!!
packet11:0x031c. This is the biggest packet bluMiU supports, because of the Bluetooth protocol restrictions that will not allow it to become bigger.
It is 796 (0x031c) bytes. This concludes the validation process of the transfer mechanism of bluMiU!!!!
Since there are many many many more bytes to be filled up, random things will follow till the payload reaches
796. Thank you very much for your patience. There.. I hope you enjoyed out time together today.. You know it seems harder and harder to
just sit back and enjoy the finer things in life.. Well, till next time.. Ta ta!!!! 0x01 0x02 0x03 0x04 0x05 0x06 ,
0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20!!!
This is the end.. 0x031c!!
  
```

**Figure 5-3 Data packet exchanging in bluMiU, using the bluMiU protocol (server to client)**

In the screenshot of Figure 5-4 12 packets that have been transmitted by the bluMiU client are shown, after they have been received by the server. The same packets and the same debugging mechanism as before (in the server to client test) were used. The server, in contrast with the client, transmits bluMiU data packets to the EXT DEV, thing that can be seen on the screenshot, where the headers can be seen as boxes [the payloads are either non printable characters for small (0x01, 0x02...) and bigger than 255 payloads (0x100, 0x200...) or printable characters for payloads that correspond with them in the normal or extended ASCII table (@ for 0x40 (64), € for 0x80 (128) and ' for 0x55 (255)]. This way the client's data packet transmission mechanism and the server's reception one were validated.



**Figure 5-4 Data packet exchanging in bluMiU, using the bluMiU protocol (client to server)**

After validating the functionality of the whole mechanism and protocol that has been designed and implemented, its efficiency should be reviewed and possible improvements of it should be commented.

### 5.3.4.2. Efficiency

Since the bluMiU platform was able to exchange data between the server and the client, the efficiency of the protocol used to transfer data should be tested.

The data transfer mechanism efficiency is determined mainly by the data rate that it can achieve in both directions. BluMiU supports speeds of up to 115.2kbps. The default data rate of 57.6kbps can be considered good for a serial transfer, since the maximum available data rate that mainstream computers support for their COM ports is 128kbps and 56kbps is the most common one for dial-up modems.

The bluMiU protocol uses data packets with a 5byte header. In spite that it seems to achieve data rates of 57.24kbps with a 57.6kbps connection and 796bytes of data payload in the packet, it doesn't. This is because the Bluetooth protocol, which is involved in the bluMiU data transferring process, needs a 9byte overhead for every data packet. With a speed of 57.6kbps (7200bytes/sec) it achieves a data rate of 56.95kbps (7118.75bytes/sec) for Bluetooth ACL data packets with payloads of 796bytes. Thus,

since the slower part of the protocol defines its speed, the data rate that bluMiU can achieve is 56.95kbps.

Another point that is vital for the efficiency of the protocol used is the ease of use it offers. BluMiU needs the external devices connected to the server and the client to be aware that it needs packets to exchange data successfully. If the external device connected to the client does not need to exchange data with the device connected to the server, but only receives data from it, the client's side does not need to know that a Bluetooth connection is employed. The server transmits to the external device connected to it bluMiU specific packets; it could provide the data only, in the same way as the client. If this case was concerned, the server would be stripped of its server duties, because the external device connected to the server couldn't be able to distinguish the client device where the data was coming from and, thus, multiple connections couldn't be supported. This is why the specified by Bluetooth SIG SPP states that *only* point to point connections are concerned. Point to multipoint connections are supported by other profiles (e.g. LAN Access Profile), which are too complicated and need a lot of hardware and software resources to be supported properly, thing that makes them unsuitable to be implemented in a weak microcontroller, such as that one used in this thesis' context.

### **5.3.4.3. Possible improvements for higher data rates**

BluMiU, as seen in the previous section, has a maximum packet payload of 796 bytes, which allows it to become very efficient, according to the speed it in can achieve.

The most evident improvement in bluMiU is to increase the data rates it can achieve by increasing the baud rates of the communication between the hardware elements of bluMiU (BT module to HOST and HOST to EXT DEV). The maximum baud rate it can be achieved this way is 230kbps, due to the limitation in the baud rate that the RS232 Line Drivers/Receivers used have. If bluMiU works in 230kbps a data rate of 227.4kbps can be reached. If a faster RS232 chip is used, baud rates of up to 460kbps could be reached, which is the maximum for the BT module. In this case data rates of up to 454.85kbps could be reached. Of course UART communication with a mainstream computer would not be possible in baud rates above 128kbps.

BluMiU can also be enhanced by increasing the maximum data payload it can accept in a data packet. For a data payload of 1KB with a 57.6kbps baud rate, bluMiU would achieve 57.1kbps. For a data payload of 2KB in the same baud rate as before, the system's data rate would reach 57.35kbps. The differences between the data rate that has been achieved (56.95kbps) and the data rates when bigger packets are utilized are not big, making the increase of the bluMiU baud rate the most important factor for achieving higher data rates.

In this chapter, all the tests that bluMiU was put through so that it would be validated as a reliable wireless data exchange mechanism were reviewed and some thoughts on its efficiency and how it can be improved were presented. In the next chapter, applications that were designed to discern the potential of bluMiU are described.

## 6. Applications

---

The validation of bluMiU meant that the requirements that were set at the beginning of the development were met. After that, applications where bluMiU would be useful were considered. BluMiU can be practically used wherever short-range wireless data exchange is needed, as long as really high bandwidth is not a prerequisite.

Since most of the Bluetooth services offer their services through the SPP, they could also be implemented using the bluMiU; it offers the same functionality. Time constraints didn't allow for the implementation of many applications and services that would display the full potential of the system. Though, file transferring through the bluMiU has been a primary target -for the designer-, since the development for bluMiU had started. So, the application that was used to demonstrate the functionality of the system was the wireless transfer of files between the client and the server.

### 6.1. *File transfer*

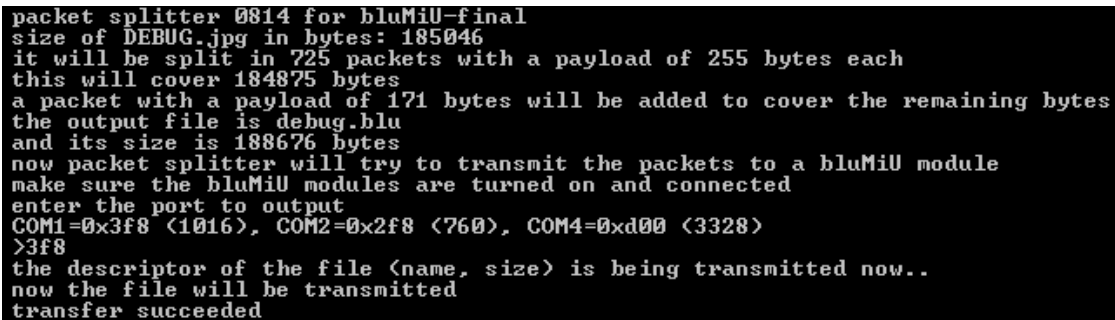
To achieve file transfer through the bluMiU an application has been implemented in C. This application contains two programs. The one is used to transmit a file to a remote device through the bluMiU and the other to receive and reconstruct the file in the remote device.

The first one (SPLITTER) splits a file to bluMiU packets and then transmits them to a remote device:

1. Takes a file (*original*) in its input and generates bluMiU packets, which carry the data contained in it
2. Creates a message, which signals file transfer initialization. This contains an indicator (0x01), the name of the *original file* and its size. It is encapsulated in an ordinary bluMiU packet
3. Generates data packets with a payload of 255bytes. The only exception is the last packet, which includes a number of data bytes equal to the remaining number of data bytes of the *original* file size modulo 255

4. Finally, it transmits all packets to the selected COM port of the computer where a bluMiU device is connected

The bluMiU device then transmits them wirelessly to the device indicated by each packet, if that device exists of course. The interface of the SPLITTER application is depicted in the screenshot below. (Figure 6-1)



```
packet splitter 0814 for bluMiU-final
size of DEBUG.jpg in bytes: 185046
it will be split in 725 packets with a payload of 255 bytes each
this will cover 184875 bytes
a packet with a payload of 171 bytes will be added to cover the remaining bytes
the output file is debug.blu
and its size is 188676 bytes
now packet splitter will try to transmit the packets to a bluMiU module
make sure the bluMiU modules are turned on and connected
enter the port to output
COM1=0x3f8 <1016>, COM2=0x2f8 <768>, COM4=0xd00 <3328>
>3f8
the descriptor of the file <name, size> is being transmitted now..
now the file will be transmitted
transfer succeeded
```

**Figure 6-1 BluMiU splitter interface**

The splitter takes two arguments; one is the filename to be transmitted, the other is the output filename; after the application finishes its job, that file will contain the bluMiU packets that have been generated by the application. The packets generated will be transmitted afterwards through the port selected to the bluMiU device that should be connected on that port.

The second program (JOINER) receives files from a client remote device:

1. It waits until a message indicating that a file transfer will start has been received, by waiting for the indicator 0x01
2. Receives the name of the file and creates it
3. Waits for the 4bytes of the long integer that will provide the *size* of the file (and, thus, the amount of bytes that the application will receive)
4. Finally, every byte that would be received next, until all the file's bytes have been received, is stored into the file created

The JOINER interface (client) follows. (Figure 6-2)

```
file re-assembler 0813 for bluMiU-final client
enter the port to input, in hex or dec format
COM1=0x3f8, COM2=0x2f8, COM4=0xd00
>d00
waiting for data from COM1740..
got descriptor header.. waiting for name length..
a file with a 9 characters long name will be received..
now waiting for the name..
the name received was debug.jpg
now waiting for the file size..
byte 0:0
byte 1:2
byte 2:210
byte 3:255
about to receive 185047 byte(s) to store in the file debug.jpg..
reception complete
```

Figure 6-2 BluMiU JOINER interface (client side)

The client's JOINER doesn't need any arguments to function correctly. It idles until a byte has been received from the port monitored. If the byte is 0x01 it is probably the beginning of a message that indicates that a file transfer will start. After that the name and the size of the file will be received and after that the file. Since this instance is targeted for a client bluMiU device, it will just receive from the port as many bytes as those defined in the message.

There is a different JOINER for the files being received from the server remote device:

1. This one waits for bluMiU packets to be received, that contain the message indicating the name and the size of the file that is going to be received
2. It creates the file and then waits for more bluMiU packets to be received
3. When the size of the bluMiU packet has been received, the data bytes contained in the packet are written to the file created
4. When enough bluMiU packets have been received, so that the size of the file has covered, the receiving of the file ends successfully

The JOINER (server) is shown in the following screenshot. (Figure 6-3)

```
file re-assembler 0817 for bluMiU-final server
enter the port to input, in hex or dec format
COM1=0x3f8, COM2=0x2f8, COM4=0xd00
>3f8
waiting for data from COM1740..
got descriptor header.. waiting for name length..
a file with a 9 characters long name will be received..
now waiting for the name..
the name received was debug.jpg
now waiting for the file size..
byte 0:0
byte 1:2
byte 2:210
byte 3:255
about to receive 185047 byte(s) to store in the file debug.jpg..
reception complete
```

Figure 6-3 BluMiU JOINER interface (server side)



The server's JOINER works exactly like the client one, except that after a message from a client (that signals the initiation of a file transfer) has been received, the server waits for bluMiU packets from the same source to be received. Multiple files from different clients can be served simultaneously, since packets that indicate their source are utilized.

Through the pair of the SPLITTER-JOINER programs (one instance of each one must be running in two different computers) files between a bluMiU server and client can be exchanged, if these two are connected to each other. The correct transfer of several files of various sizes from the server to the client and vice versa validated the correct implementation of the application and made sure that the bluMiU system could support large file transfers. By being successful in the task of transferring large files, it became obvious that the system could support every kind of data streams.

The JOINER program can function as a file server. It just waits for files to be transmitted to it. It is not a fully functional file server and cannot replace the File Transfer Profile specified by the BTspec, since it is only able to accept files and does not offer browsing, downloading or manipulating files and folders capabilities. [7] Through the bluMiU protocol, these services can be provided in the same abstraction level as the JOINER and the SPLITTER.

## 7. Conclusions and Future Work

---

### 7.1. *Conclusions*

A system capable of reliable wireless data transfers, which supports high baud rates and utilizes the BlueTooth technology to achieve data exchange was designed and built in this thesis. The whole design has been embedded in a low-cost and low-power microcontroller that lacked large memory (1KB of RAM) and rich hardware resources.

BlueTooth technology has been explored in the areas where data transfer was concerned and, since the Serial Port Profile (SPP) that has been specified by the BlueTooth Special Interest Group (SIG) in the BlueTooth specification (BT spec) 1.2 proved to be resource demanding and couldn't cover the requirements of the system, a differentiated protocol that supports point-to-multipoint connections has been designed, implemented and tested. The protocol that was designed proved through the tests to be highly efficient, in spite the memory limitations of the system. The whole design was based on BT spec 1.2 and can support every BlueTooth module that complies with that BT spec<sup>1</sup>.

A BlueTooth device is usually able to find devices and accept connection requests from remote devices. The system implemented in this thesis adopted a client-server architecture. The server device retained the BlueTooth functions that enable it to be found, respond correctly to connection requests from multiple BlueTooth devices and exchange data with every one of them. Correspondingly, the client device retained the BlueTooth functions that enable it to find and identify BlueTooth devices. It also has the capability to establish a connection with one of the devices it found and finally exchange

---

<sup>1</sup> Since bluMiU was designed to work with a BT spec 1.0b qualified BlueTooth module, it should work with every BlueTooth module available.

data with it. By sharing the BlueTooth characteristics, the microcontrollers used were relieved from about half the load in hardware resources they had.

The applications that were developed showed the potential of bluMiU as a full featured wireless data transferring system capable of high baud rates.

### **7.2. Future work**

BluMiU is a complete data exchange system. The improvements that could be made to the design deal with increasing its speed and adding services in a higher level, since the system works infallibly. Some improvements to the existing design and some new applications that this design would be useful would be the following:

- I. The most obvious improvement in the design would be to improve its data rate, as discussed in chapter 5.3.4.3 (page 79). The BT module and the microcontroller used both support high baud rates. The highest possible the specific BT module can achieve is 460.8kbps, while the microcontroller can achieve 912.6kbps. In such high speeds the BT module's UART buffers might fill before the packets in them will be sent over the air, so flow and congestion control must also be implemented. It is possible that data rates above 128kbps may need another interface protocol (than UART) for the connection between computers and bluMiU, because computers do not easily support higher UART speeds. The implementation of a USB core seems to be a viable solution.
- II. Most BlueTooth profiles can be implemented through the bluMiU data exchange protocol. Profiles than can be implemented are:
  1. LAN Access (LAP)
  2. Generic Object Exchange (GOEP)
  3. Object Push (OPP)
  4. Synchronization (SP)
  5. File Transfer (FTP)

A demonstration application that resembles the File Transfer Profile has been developed in the context of this thesis.

- III. The computer-bluMiU interface can be improved to enable the fully automatic function of the bluMiU by developing a special driver for it and adding to the

bluMiU design more reports of its status to the computer or the external device connected to it.

- IV. Another BT module can be utilized. More connections can be supported and higher data rates can be achieved. The various sleep modes that the BT spec 1.2 defines can be used in order to support more than 7 connected devices. The Enhanced Data Rate that is about to appear in the market will enable even higher data rates and broadcasting (sleeping devices also accept broadcast messages) can be also used in order to achieve better results in multiple connections.
- V. Another microcontroller or an FPGA can be used to replace or work along the one used. This would eliminate the need for a computer and/or a human presence and will provide more memory. This way the development of a new generation of bluMiU could be initiated, were the server and the client will be merged into one device, allowing bluMiU devices to form a chain of connected devices that will be able to function as a network. Larger memory can help to the development of Bluetooth network routers that will contain routing tables and thus implement routing algorithms. This can help to expand the range of the Bluetooth network to more than 10m by having Bluetooth routers every 10m that will forward the data packets to the correct path.
- VI. A larger memory would help bluMiU to become completely transparent to the top-level applications that will be able to see it as a simple UART cable, instead of a packet exchange mechanism, as long as multiple connections are not concerned.
- VII. The bluMiU can also be used for FPGA programming by combining it with the Hardware Programmer & Tester (HPT). HPT is a generic FPGA programmer board (using a UART I/F) that has been developed by Dionysios Efstathiou in the MHL, ECED, TUC [14], which utilizes the ReRun instruction set and Application Program Interface (API) created by Thomas Kyriakidis in MHL, ECED, TUC [15]. The bluMiU interface can be used to program many generic autonomous FPGA-Bluetooth systems (robots) that belong to a community and collaborate through their communication with a server. The server will control, synchronize and re-program the robots' FPGAs so that they would be able to accomplish different tasks. The tasks would depend on the information it collects from them and their

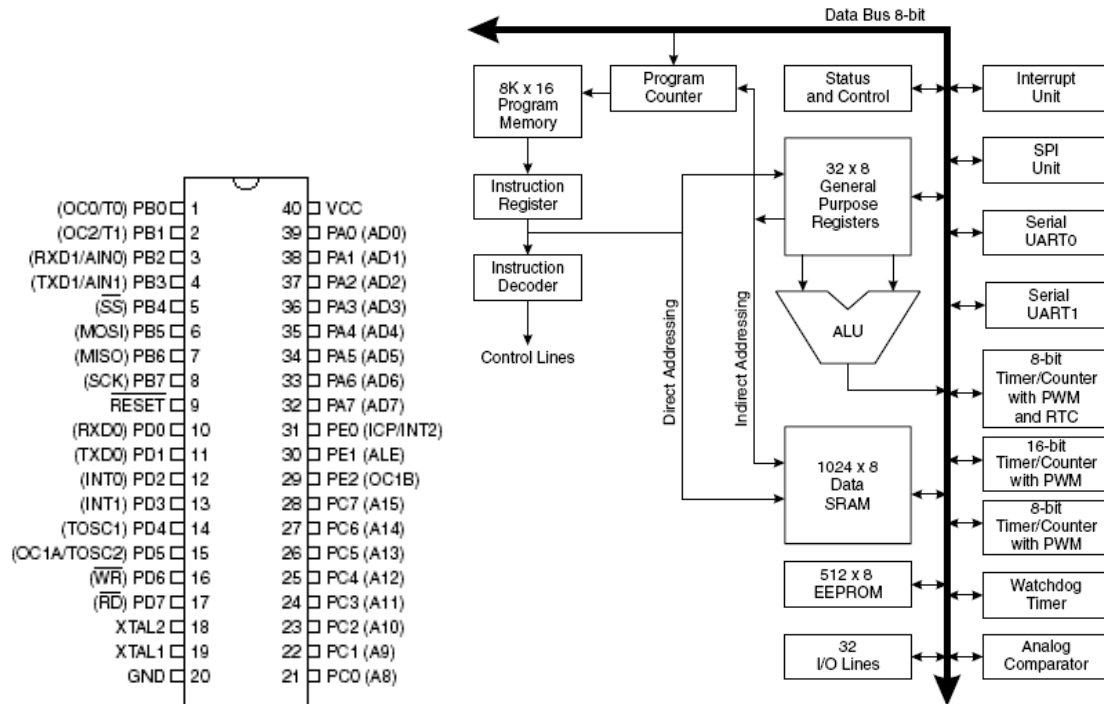
## **Microprocessor & Hardware Laboratory**

neighbors. When all the tasks are completed, the system would have accomplished a complicated assignment.

# Appendix A

## ATmega161 AVR

The microcontroller (mC) used as the HOST (controls all the hardware components and synchronizes their operation) in bluMiU is the ATmega161 AVR by Atmel.



**Figure 0-1 ATmega161 PIN configuration and architecture**

The mC used in bluMiU is an Atmel ATmega161 AVR. The ATmega161 is pin compatible with AT90S8515, which was used in the BlueAppIE architecture, thing that helped the transition from that architecture to bluMiU by making the latter a direct evolution of BlueAppIE. The ATmega161 pin configuration and architecture are depicted in Figure 0-1. The ATmega161 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing instructions in a single clock cycle, the ATmega161 achieves throughputs approaching 1 MIPS per MHz. The AVR core combines an instruction set, which contains 130 instructions, with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The proposed frequency of operation for the mC is 8MHz.

Although an XTAL with a frequency of 11.0592MHz is used in bluMiU. This frequency has certain advantages over all the others: (1) It is the highest permitted by the mC's datasheet (providing a higher performance) and (2) by using that frequency a 0% error rate can be achieved with the mC's UART. [14] The ATmega161 provides the following features:

- 16K bytes of In-System or Self-programmable Flash
- 512 bytes EEPROM
- 1K byte of SRAM
- 35 general purpose I/O lines (4x8bit and 1x3bit)
- 32 general purpose working registers
- Real-time Counter
- 3 flexible Timer/Counters with Compare modes (2x8bit and 1x16bit)
- 18 internal and 3 external interrupts
- 2 programmable serial UARTs
- programmable Watchdog Timer with internal Oscillator
- 1 SPI serial port
- 1 2-cycle multiplier
- 3 software-selectable power saving modes:
  - I. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port and interrupt system to continue functioning
  - II. The Power-down mode saves the register and SRAM contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset
  - III. In Power-save mode, the timer Oscillator continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping

The On-chip Flash Program memory can be reprogrammed using the self-programming capability through the Boot Block and an ISP through the SPI port, or by using a conventional non-volatile Memory programmer. [14]

## Microprocessor & Hardware Laboratory

Microcontroller resources used by bluMiU<sup>1</sup>:

- ✓ 11.6KB and 8.97KB of the 16KB Flash for the client and server modules respectively
- ✓  $(671+n*10)$ Bytes and  $(571+n*10)$ Bytes of the 1KB SRAM for the client and server respectively ( $n*10$ Bytes are stored in the RAM for  $n$  interrupts triggering one after the other because every interrupt needs 10Bytes to be stacked in the RAM, so that a previous interrupt can be resumed)
- ✓ 28 of the 35 I/O lines. 22 are used for output and 6 for input
- ✓ All the general purpose registers of the mC for the client, while 30 of the 32 general purpose registers for the server
- ✓ 1 of the 2 16bit timers
- ✓ 1 of the 3 external interrupts
- ✓ Both UARTs
- ✓ Idle power saving mode is used

---

<sup>1</sup> BluMiU client & server version used: 0715



# Appendix B

## *The BT module*

The BT modules used are the hardware part of the Bluetooth Application Tool Kit, LZT 108 4123 R2A, developed by Teleca Comtec. The hardware consists of a two-layer printed circuit board equipped with a UART buffer, a voltage regulator, a USB connector, an inverted-F antenna, a few other passive components and the Bluetooth ROK101 008 module of Ericsson (Figure 0-1). This BT Module includes the Ericsson Baseband device, a Flash Memory and the Ericsson Radio Module device. Qualification for the Tool Kit is based upon a declaration of compliance with the Bluetooth Specification 1.0b plus critical errata. [15]

The BT module in the BlueApplE design had as a power source a USB cable that connected it with a computer. Using up a USB port just to provide power is clearly a waste of resources. In the bluMiU design the board that includes the HOST and all the hardware components is the one that provides power to the module. The USB power supply can be changed by a power source that is connected to the jumper area of the board (Figure 0-1) in connectors 1(Vin 5V nominal) and 10(GND) that fulfills the following requirements:

- Supply voltage: min +4.4 V, max +5.25 V connected to Jumper area pin 1 (relative GND pin 10)
- Minimum supply current: 100 mA [15]

A description of the ROK 101 008 and its functions will follow.

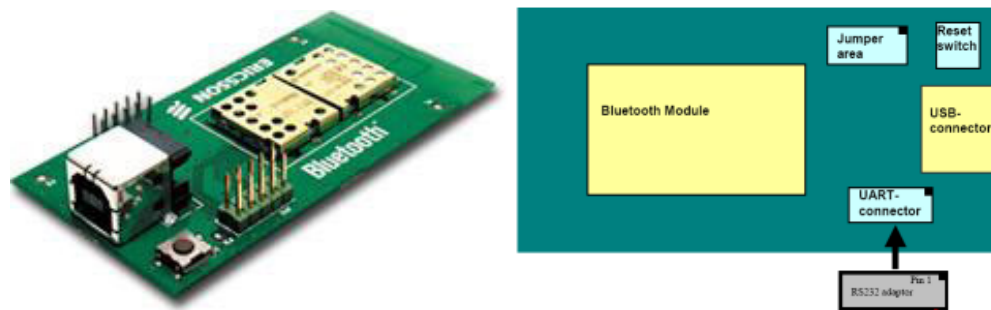


Figure 0-1 LZT 108 4123 R2A

ROK 101 008 is a short-range module for implementing Bluetooth functionality into various electronic devices. It is compliant with BT spec 1.1, is a Class 2 Bluetooth Module (0 dBm) and is type-approved. The module consists of three major parts; a baseband controller, a flash memory, and a radio that operates in the globally available 2.4–2.5 GHz free ISM band. The baseband controller is an ARM7-Thumb based chip that controls the operation of the radio transceiver via the UART interface. The module's flash memory includes firmware for the HCI and the LM, which were discussed in chapter 3.1 (page 19). Both data and voice transmission is supported by the module. Communication between the module and the Host is carried out via UART and/or PCM interface. The UART implemented on the module is an industry standard 16C450 and supports the following baud rates: 300, 600, 900, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 and 460800 bits/s. 128 byte FIFOs are associated with the UART. [16] The default setting for the UART speed is 57.6Kbps and can be changed by sending to it an Ericsson specific HCI command. HCI command packets sent to it should have 0x01 as packet indicator (a byte that must be transmitted immediately before any HCI packet), events transmitted by it have 0x04 as packet indicator and HCI ACL packets are interchanged with the packet indicator 0x02. It supports only a point-to-point connection, and thus disallowing the bluMiU design to support point-to-multipoint connections. Newer versions of this same module offer point-to-multipoint connections, without the slightest change in their architecture or the way they communicate with the HOST, which is exactly the reason why bluMiU is said to support point-to-multipoint connections too. [17]

# Acronym List

---

## A

**ACL:** Asynchronous Connection-Less

**ACL-C:** ACL-Control

**ALU:** Arithmetic Logic Unit

**API:** Application Program(ming) Interface

## B

**BD ADDR:** Bluetooth Device Address

**BlueAppIE:** Bluetooth Applications Environment

**BlueBridge:** Bluetooth Data Bridge

**BluMiU:** Bluetooth Multi-UART

**BT module:** Bluetooth module

**BTspec:** Bluetooth specification

**BufLen:** Buffer Length (bluMiU flag indicating the number of stored bytes in the buffer)

**BUFOVF:** Buffer Overflow (bluMiU flag indicating that the buffer is full)

## C

**CAGR:** Compound Annual Growth Rate

**CID:** Channel Identity

**CLK:** Clock

**CMOS:** Complementary Metal Oxide Semiconductor

**COM:** Communication

**CPU:** Central Processing Unit

**CRC:** Cyclic Redundancy Check

## D

**DH:** Data - High Rate

**DM:** Data - Medium Rate

## **E**

**ECED:** Electronics & Computer Engineering Department

**EDR:** Enhanced Data Rate

**EEPROM:** Electrically Erasable Programmable Read-Only Memory

**EI:** Enable Input

**EXT DEV:** External Device

## **F**

**FEC:** Forward Error Correction

**FHSS:** Frequency Hopping Spread Spectrum

**FIFO:** First In, First Out

**FLAGREG:** Flag Register (bluMiU register containing various flags)

**FPGA:** Field-Programmable Gate Array

**FTP:** File Transfer Profile

## **G**

**GFSK:** Gaussian Frequency-Shift Keying

**GND:** Ground

**GOEP:** Generic Object Exchange Profile

**GS:** Group Signal

## **H**

**HCI:** Host Controller Interface

**HPT:** Hardware Programmer & Tester

## **I**

**I/F:** Interface

**I/O:** Input/Output

**IC:** Integrated Circuit

## **Microprocessor & Hardware Laboratory**

**IEEE:** Institute of Electrical & Electronic Engineers

**IEEE-SA:** IEEE - Standards Association

**INT:** Interrupt

**IP:** Internet Protocol

**IR:** InfraRed

**IrDA:** Infrared Data Association

**ISM:** Industrial, Scientific & Medical

**ISR:** Interrupt Service Routine

### **L**

**L2CAP:** Logical Link Control & Adaptation Protocol

**LAN:** Local Area Network

**LAP:** LAN Access Profile

**LED:** Light Emitting Diode

**LM:** Link Manager

**LMP:** Link Manager Protocol

### **M**

**MAC:** Media Access Control

**mC:** microcontroller

**MHL:** Microprocessor & Hardware Laboratory

### **O**

**OBEX:** Object Exchange Protocol

**OCF:** Opcode Command Field

**OGF:** Opcode Group Field

**Opcode:** Operation code

**OPP:** Object Push Profile

### **P**

**PAN:** Personal Area Network

**PCM:** Pulse Code Modulation

**PDA:** Personal Digital Assistant

## Q

**QoS:** Quality of Service

## R

**RAM:** Random Access Memory

**RdBuf:** Read Buffer (bluMiU flag indicating where the next byte will be read from)

**RF:** Radio Frequency

**RISC:** Reduced Instruction Set Computer

**RS232:** Recommended Standard 232

**RX:** Receive

**RXD:** Received Data

## S

**SCO:** Synchronous Connection-Oriented

**SD:** Secure Digital

**SIG:** Special Interest Group

**SP:** Synchronization Profile

**SPI:** Serial Peripheral Interface

**SPP:** Serial Port Profile

**SRAM:** Static RAM

## T

**TDD:** Time Division Duplex

**TTL:** Transistor-Transistor Logic

**TUC:** Technical University of Crete

**TX:** Transmit

**TXD:** Transmit Data

## U

**UART:** **U**niversal **A**synchronous **R**eceiver-**T**ransmitter (also serves as a bluMiU flag indicating that the UART is not in the process of sending a byte)

**UCR:** **U**ART **C**ontrol **R**egister

**USB:** **U**niversal **S**erial **B**us

**USR:** **U**ART **S**tatus **R**egister

**UxRX:** **U**ART **x** (x can be 0 or 1) **R**eceive Complete ISRs (bluMiU)

**UxTX:** **U**ART **x** (x can be 0 or 1) **T**ransmit Complete ISRs (bluMiU)

## W

**WiFi:** **W**ireless **F**idelity

**WLAN:** **W**ireless **L**AN

**WMAN:** **W**ireless **M**etropolitan **A**rea **N**etwork

**WrBuf:** **W**rite **B**uffer (bluMiU flag indicating the buffer location to store the next byte)

## X

**XTAL:** Crystal

# References

---

## *Literature*

- [1] Brent A. Miller - Chatschik Bisdikian, BlueTooth Revealed: The insider's guide to an open specification for global wireless Communications, Prentice Hall PTR, 2001
- [3] Specification of the BlueTooth System, Covered Core Package version 1.2
- [4] Ericsson Technology Licensing AB, Bluetooth Beginner's Guide
- [5] Christos Strydis, Diploma Thesis: Interface and Wireless Embedded Applications of Bluetooth, based on Microcontrollers, Technical University of Crete, Electronics & Computer Engineering Department, Microprocessor & Hardware Laboratory, 2003
- [7] Specification of the BlueTooth System, Profile Book version 1.1
- [13] James F. Kurose - Keith W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, 2<sup>nd</sup> edition, Pearson Education-Addison Wesley, 2003
- [14] Dionysios Efstathiou, Diploma Thesis: Design and Implementation of a Vendor-Independent Universal Programmer for FPGA Technology, Technical University of Crete, Electronics & Computer Engineering Department, Microprocessor & Hardware Laboratory, 2002
- [15] Thomas Kyriakidis, Diploma Thesis: Development of a language and Universal Run-Time Environment for FPGA programming, Technical University of Crete, Electronics & Computer Engineering Department, Microprocessor & Hardware Laboratory, 2002
- [16] Atmel ATmega161(L) AVR datasheet, Rev. 1228C-AVR-08/02, 2002 Atmel Corporation
- [17] Getting Started - Bluetooth Application and Training Tool Kit, Ericsson Technology Licensing AB, 2001
- [18] Ericsson ROK 101 008 Bluetooth PtP Module datasheet

## *Internet Resources*

- [2] The Official BlueTooth Membership Site (<https://www.bluetooth.org/>)
- [6] Bluetooth 2004: Poised for the Mainstream (#IN0401211MI), In-Stat/MDR, 2004
- [8] Marc de Courville, HIPERLAN: an OFDM solution granting compatibility between next G of high rate WPANs and WLANs, Proposal to the IEEE P802.15 Working Group, 2000
- [9] Institute of Electrical & Electronics Engineers website (<http://www.ieee.org>)
- [10] David Coursey, Bluetooth vs. WiFi: Why it's NOT a death match, <http://reviews-zdnet.com.com>, 2002
- [11] IEEE announcement for the approval of IEEE 802.15.1 Standard for Wireless Personal Area Networks Adapted from the BlueTooth Specification:  
<http://standards.ieee.org/announcements/802151app.html>
- [12] Dave Suvak, IrDA & Bluetooth: A Complementary Comparison, <http://www.dpi.net.ir>, 2003
- [19] Teleca Comtec AB Site (<http://www.comtec.teleca.se>)