Microprocessor & Hardware Laboratory

Technical University of Crete

Department of Electronic and Computer Engineering

Diploma Thesis

Programming of Reconfigurable Logic Using the Bluetooth Protocol

Konstantinos Kazakos

Supervising Professor: Professor Apostolos Dollas

Thesis Committee: Professor Michalis Paterakis
Assistant Professor Dionysios Pnevmatikatos

June 2006

"The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to him. Therefore all progress depends on the unreasonable."

Unknown

Dedicated to my family

I would like to thank

Professor Apostolos Dollas for his invaluable support throughout the whole duration of this thesis, for the knowledge he has offered me all these years, for his guidance and toleration and for giving me the opportunity to be a member of Microprocessor and Hardware Laboratory (MHL).

The thesis committee, Professor Michalis Paterakis and Assistant Professor Dionysios Pnevmatikatos for their contribution to this thesis.

Dionysios Efstathiou for his unbelievable help and Elias Politarhos as well as Christos Strydis for their guidance.

Markos Kimionis, member of the Technical Staff of the MHL for his support regarding technical matters, for the unparalleled patience he has shown and for the many hours he has spent helping me.

Dr. Brian von Herzen of Rapid Prototypes Inc. (www.FPGA.com) for his useful suggestions and contributions to this thesis (including the Parrotfish project name).

All the undergraduate and graduate students for their valuable help.

The Teleca Comtec Company for its kind donation of the new Bluetooth modules (www.teleca.com).

The Crinno Unistep program for their help on matters of financial support.

And last, but not least, I would like to thank my parents and my friends for being there to support me.

Contents

1. Introduction

1.1. The Bluetooth® System and the Parrotfish project

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.

In early 1998, the Bluetooth Special Interest Group (SIG) was formed by Ericsson, IBM, Intel, Nokia and Toshiba. Now (June 2006) 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 (promoters, associates[1] & adopters[2]). 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

---

[1] Associate members have the opportunity to work with other Associate and Promoter companies on enhancements to the Bluetooth specification

[2] Adopter members just make use of Bluetooth technology

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.

Today, 8 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: 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[3] (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

[3] 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

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-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).

At the same time, reconfigurable resources can provide substantial field programmable capability in the wireless autonomous nodes of an ad-hoc sensor network. The driving problem arises from the necessity to have exchange of (re)configuration and partial reconfiguration bit streams as well as data in a network that is inherently unstable (even with respect to its topology). The problems are compounded by the need for low cost solutions and support for multi-vendor systems. Thus, there is a need to reconfigure or partially reconfigure individual nodes in order to alter system behavior, or circumvent non-fatal errors. The Parrotfish project (named so due to individual Parrotfish changing gender under group dynamics) is a low cost, distributed environment for (partial) reconfiguration of distributed field programmable systems, e.g. sensor networks.

This thesis' main target is to integrate the Bluetooth protocol with the Reconfigurable hardware through the Parrotfish project. It proposes a system that can work as an errorless and efficient data communication medium in a Parrotfish node for low cost re/configuration of an FPGA in an ad hoc network.

1.2. Thesis Stimulus, Scope & Contributions

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[4]. This fact in conjunction with the kind donation of Teleca Comtec of the three point to multipoint Bluetooth modules that were used in this thesis and the development of an embedded applications platform of Bluetooth in the MHL by Christos Strydis [7] and Elias Politarhos [8], as well as the creation of an vendor-independent universal FPGA programmer by Dionysios Efstathiou [9], gave the inspiration for this thesis. In this thesis a third generation embedded application platform (**Blu**etooth **Re**configurable **Run** Time Environment) has been developed. This platform works as a subsystem in the Parrotfish project providing efficient data communication between a master and two or more slaves and also an effective packet exchanging protocol suitable for very low processing power microcontrollers (microcontroller used: Atmel AVR ATmega162). The ultimate goal of this thesis is to wirelessly program an FPGA through the Parrotfish project, but the specific design can be used wherever wireless transfer of data is needed and resources are truly limited.

The following contributions were made in this thesis' context:

1) *Piconet-based master-slave architecture*.
2) *Point to multipoint connections*
3) *Addressing protocol implemented and point to multipoint algorithm executed*
4) *Dynamic HCI command configuration is supported*.
5) *Error handling for efficient data transport implemented*.
6) *The upgrade of commands*
7) *Merging of master and slave in one device*
8) *Payload data packets are big enough to support the data send to the FPGA.*
9) *software-interrupt based multiprogramming*
10) efficient *Connectionless data transport used for FPGA re/configuration*

[4] 7 active slaves & 255 parked slaves are supported by BTspec 2 and all versions of BTspec before it

1.3. Thesis organization

Chapter 2: The relative research on the Bluetooth protocol. The Bluetooth protocol is analyzed and thoroughly examined.

Chapter 3: It reviews the existing design pros and cons and gives an overview of the new architecture. It, also, examines briefly the HPT architecture.

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

Chapter 5: The test procedures of the components of the system and the validation of the overall system are presented.

Chapter 6: 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.

Appendix A, B, C: These three appendixes include the description of the Host used in this thesis (ATMEGA162), of the Bluetooth Module and a manual as well as the PCBs of the BluReRun nodes and functions.

2. Relative Research

2.1 Bluetooth Introduction

During the last couple of years the Bluetooth devices and those that support the Bluetooth protocol have entered the true mainstream and have become widespread. With Mobile phones, Personal Digital Assistants (PDAs), and headsets making significant progression over the last years, the wireless market beginning to make an impact, and Personal Mobile Gateway[5] products emerging, shipments of Bluetooth-enabled manufactured equipment will experience a 40% growth rate of between 2006 and 2008[1] (Figure 2-1).

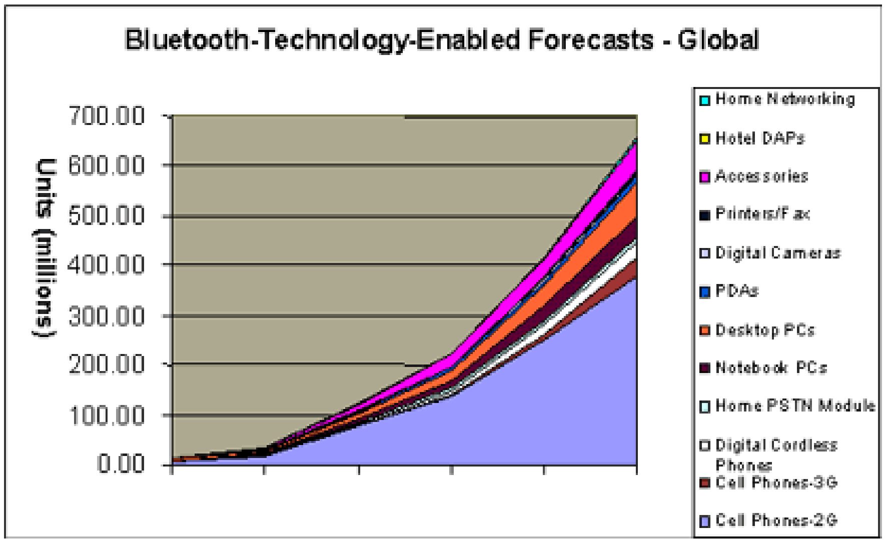


Figure 2-1: Bluetooth modules forecast (Units in millions) (In-Stat/MDR, 4/05, [1])

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 simple market and wanting to be in the

[5] 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/)

cutting edge of technology, as far as communications are concerned, is using the Bluetooth technology, because of the Bluetooth popularity, thanks to the hard work of the Bluetooth Special Interest Group on marketing [2]. It is essential for products targeting to a mainstream market to be easy to use and to have a user friendly interface. Hence worth, these products tend to waste resources on various services that the Bluetooth protocol offers, something which must be taken under consideration from the designer as well as the software developer.

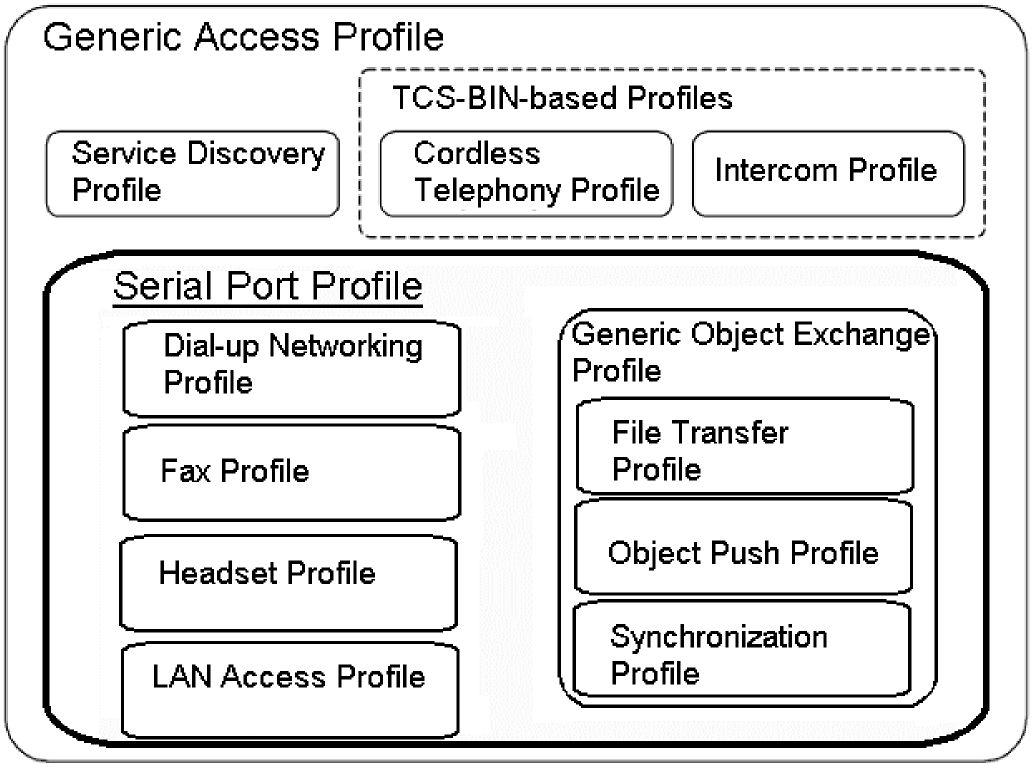


Figure 2-2: The importance of Serial Port Profile (BTspec 2 [3])

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. Actually the SPP is nothing more other than an emulation of the connection between two or more Bluetooth devices. This is the profile on which almost every other service provided by the Bluetooth protocol is based. [3] (Figure 2-2). As it will be thoroughly examined in the subsequent chapters, the main scope of this thesis interferes with the Serial Profile Protocol of Bluetooth.

2.2 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 communicating through electromagnetic waves in the infrared light (IR) and radio frequency (RF) areas --RF is the technology that Bluetooth uses as it will be seen in section 2.3. A brief overview of these two technologies will be presented in the following sections, along with a comparison to Bluetooth, where applicable.

2.2.1 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 Personal Area Network (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[6] 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. For instance, in Europe and in the United States the 900MHz, 2.4GHz (ISM) and 5GHz bands do not require license for a technology to use them and, thus, it can be easily presumed that all RF technologies for

[6] NTIA Manual of Regulations & Procedures for Federal Radio Frequency Management: http://www.ntia.doc.gov/osmhome/redbook/redbook.html

Personal Area Networks use one of these bands to function (the 5GHz band has been proposed for the Institute of Electrical & Electronics Engineers (IEEE) HIPERPAN standard [5]). 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). [3] 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[7]. 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

[7] Bluetooth FHSS is explained in chapter 2.3.1

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:

- 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 Local Area Network. [4]

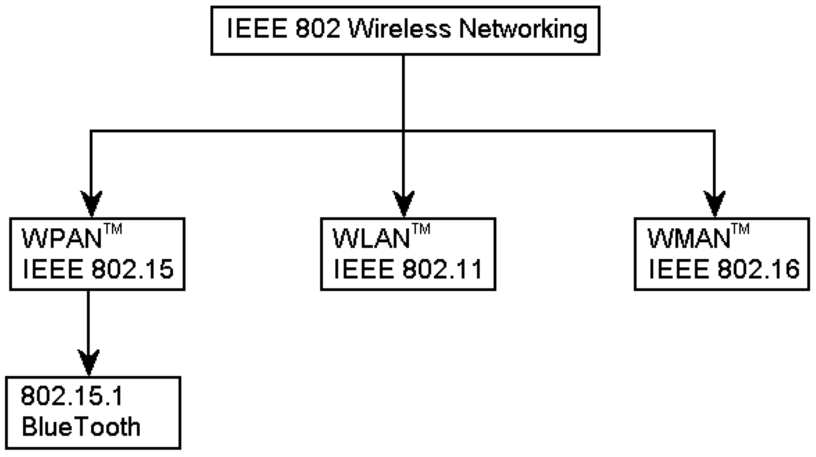


Figure 2-3. IEEE wireless networking standards

Bluetooth (as IEEE 802.15.1™) was approved 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). 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).

2.2.2 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)[8], which defines the hardware and software protocols for wireless communication intended to promote interoperable applications, 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 costumer to select the technology that fits his needs. The two technologies have their pros and cons, which help the costumer 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)

[8] http://www.irda.org

- 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 []

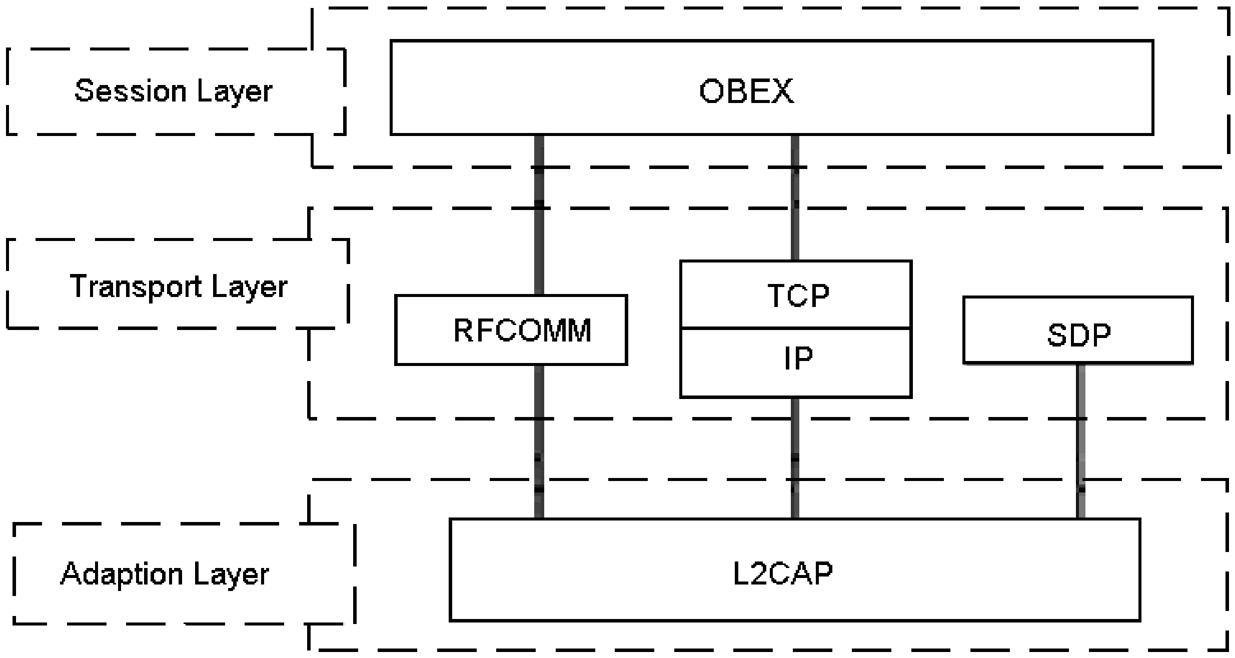


Figure 2-4. IrDA OBEX how it is reused in Bluetooth (BTspec 2 [5])

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[9] and specifically the infrared Object Exchange Protocol (OBEX) was identified as the design direction of the Bluetooth SIG early in the specification's development.

[9] IrDA had been already developed when Bluetooth research started

The purpose of the OBEX protocol is to enable the exchange of data objects and files. (Figure 2-4).

At the following section the Bluetooth Protocol and the protocol's levels and functionalities are examined thoroughly.

2.3 The Bluetooth protocol

The Bluetooth module used in this thesis is a point to multipoint device (further information are given to Appendix C) and can be considered as a black box that can be "manipulated" by a central microcontroller unit (HOST) for each node.

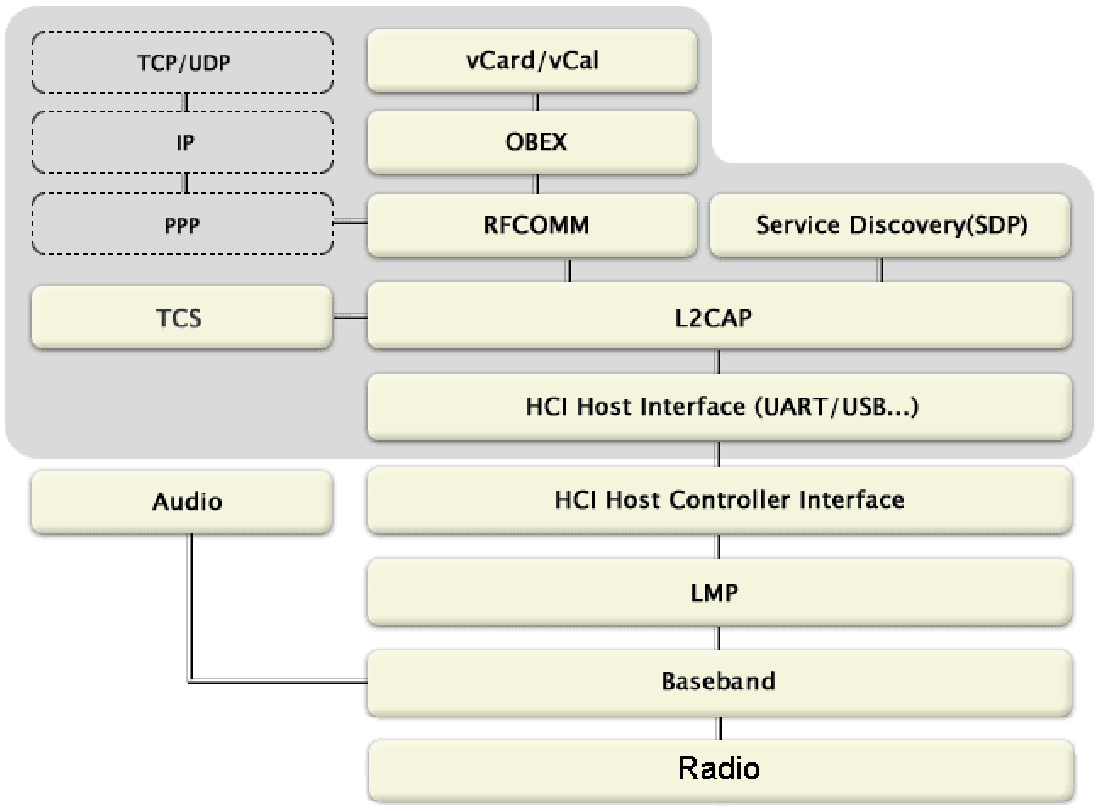


Figure 2-5: Bluetooth Protocol Stack version 2 (BT spec 2 [4])

If the parts of the Bluetooth protocol that are embedded in the Bluetooth 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 Bluetooth specification 2 which is the last specification hyper-protocol introduced by the Bluetooth SIG (Special Interest Group). The Bluetooth module used is a Bluetooth Controller, Bluetooth spec 2 compliant, which specifies the protocol stack illustrated in Figure 2-5. The layers of the Bluetooth protocol stack that are embedded in the lower levels of the Bluetooth module that is used are:

- The Radio, where the physical channel resides and data is transmitted between Bluetooth devices [5]
- The Baseband, a link controller which carries out the Baseband protocols and other low-level link routines [5]
- The Link Manager Protocol (LMP), which is used for link set-up and control [5] and
- The Host Controller Interface (HCI), which provides a command interface to the Baseband controller and link manager, and access to configuration parameters [5]

Bluetooth protocol stack layers above HCI are embedded in the microcontroller used. No SCO (Synchronous Connection Oriented) channels are used since such a function is not in this thesis’ scopes. The lower layers significant for this thesis completion and proper function are thoroughly examined in the following sections.

2.3.1 Bluetooth Radio

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 [5] as shown in the following formula: **[Frequency (MHz)] = 2402+k,** where **k = {0, 1 …, m-1}** and **m=79** (until 2004 Spain, France and Japan had m=23, because their ISM band was narrower; but this has changed in 2006 or is in the progress of changing). 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 625μs and it is used until a next frequency is selected. The time of 625μs 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). [5]

Symbol rate is 1Mbps with the use of a GFSK[10] 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. 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 [5]. TDD is explained in more detail in section 2.3.2.3.

2.3.2 Baseband

2.3.2.1 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

[10] The signal passes through a Gaussian filter and then goes through an FSK modulator (Frequency-Shift Keying)

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 2-6). The maximum number of devices that can be active in a piconet is 7 [5]. 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 hoping 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.

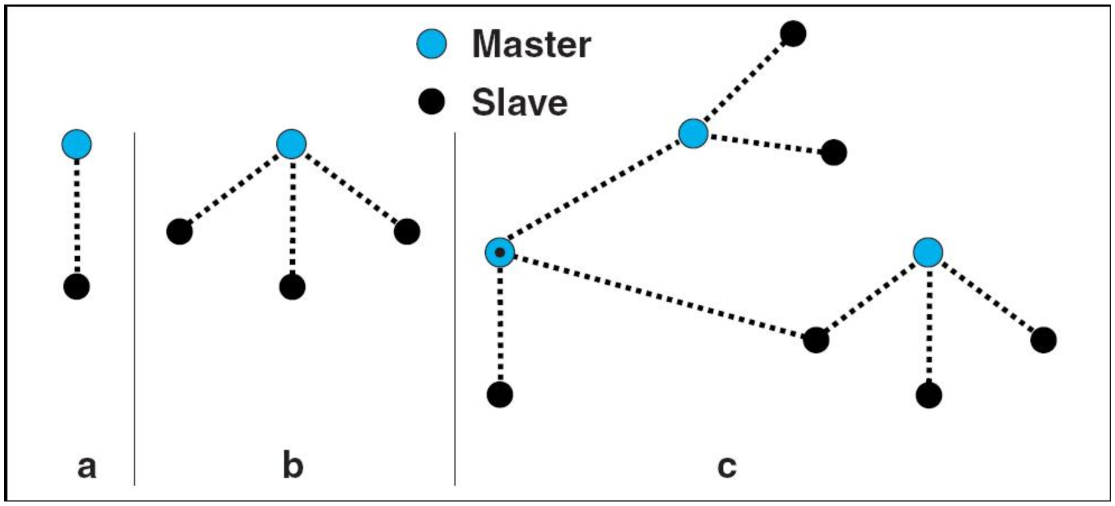


Figure 2-6: Point to point and point to multipoint connections

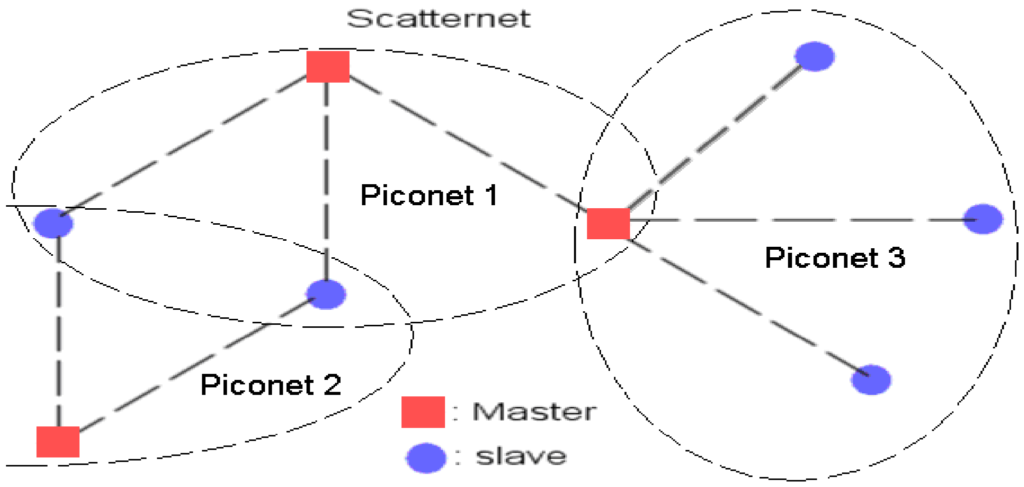


Figure 2-7: Scatternet

Piconets that have common devices are called a scatternet. (Figure 2-7). 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 [5]. 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.

2.3.2.2 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[11]. 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 the channel. Once a piconet has been established, Master-slave roles may be exchanged [5].

2.3.2.3 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 μs 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 2-8 below. The

[11] 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

packet start shall be aligned with the slot start and one can extend over up to five time slots [5].

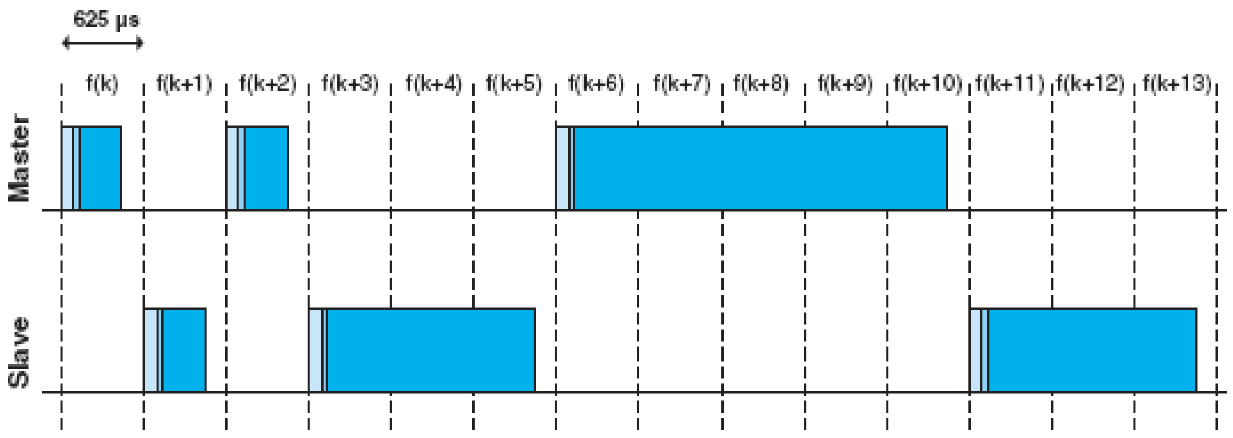


Figure 2-8: TDD in Bluetooth

2.3.2.4 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 2-9. 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 [5]. From the Standby state the device can move to Page or Inquiry state.
- *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 [5].

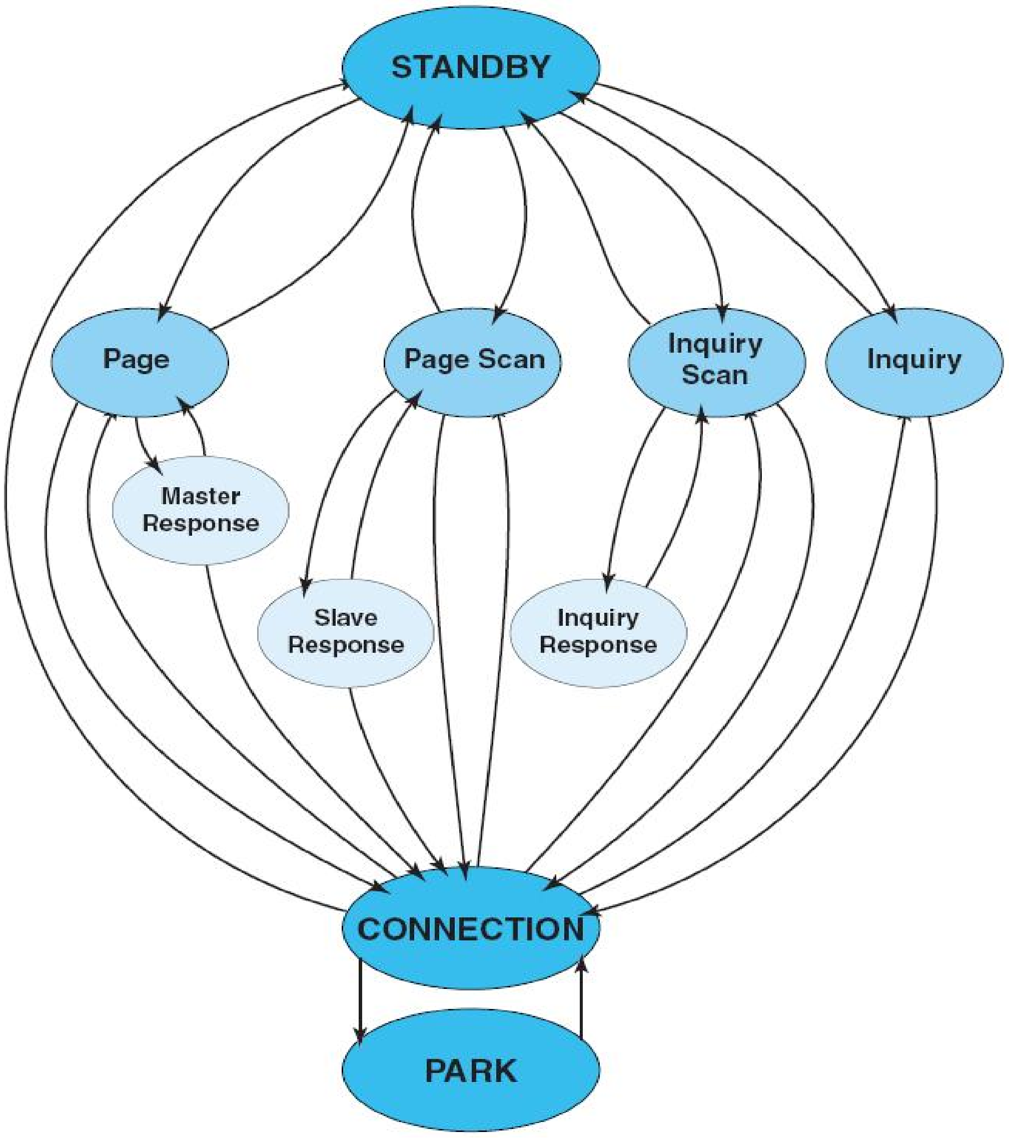


Figure 2-9: Bluetooth operational modes (states)

- *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 [5]. As Figure 2-9 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.

2.3.2.5 Packet types

The general packet type is shown in Figure 2-10. 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.

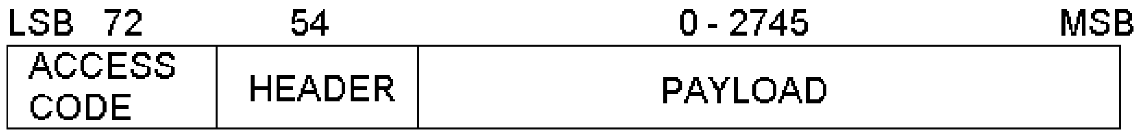


Figure 2-10: Bluetooth Baseband generic packet (Bluetooth spec 2 [5])

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=127bits, maximum 72+54+2745=2871bits)

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 [5]. 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).

2.3.3 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 [5]. 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. [6] There are three modes that can be used to reduce power consumption: sniff, hold and park (examined below)
3. Quality of Service 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 [5]. 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 [5].

2.3.3.1 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 Tsniff. (Figure 2-11). To enter sniff mode, the Master or slave issue a sniff command via the LM protocol. This message includes the sniff interval Tsniff [5].

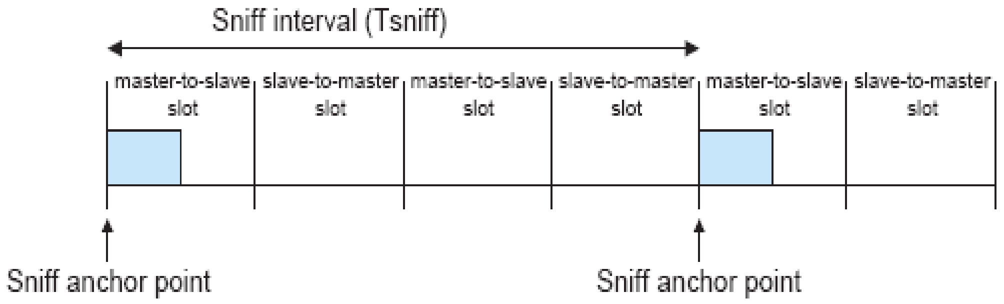


Figure 2-11: Sniff Anchor Points (Bluetooth spec 2[5])

2.3.3.2 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.

2.3.3.3 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.

2.3.4 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 2-12). 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.

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

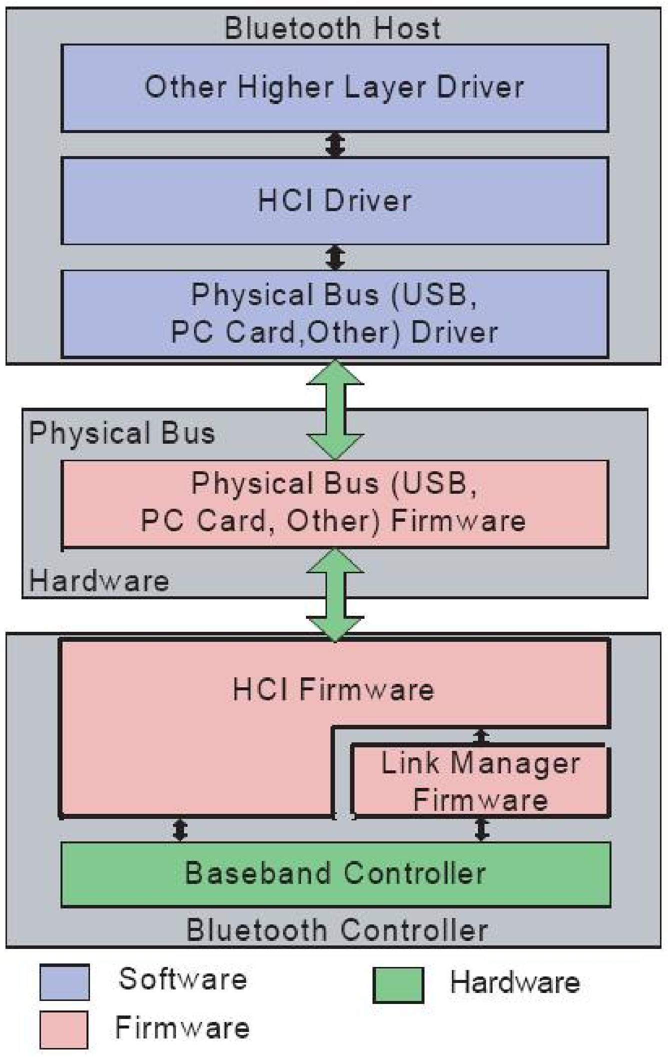


Figure 2-12: The HCI transport layer (Bluetooth spec 2 [5])

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 2-13. 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 [5]. 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.

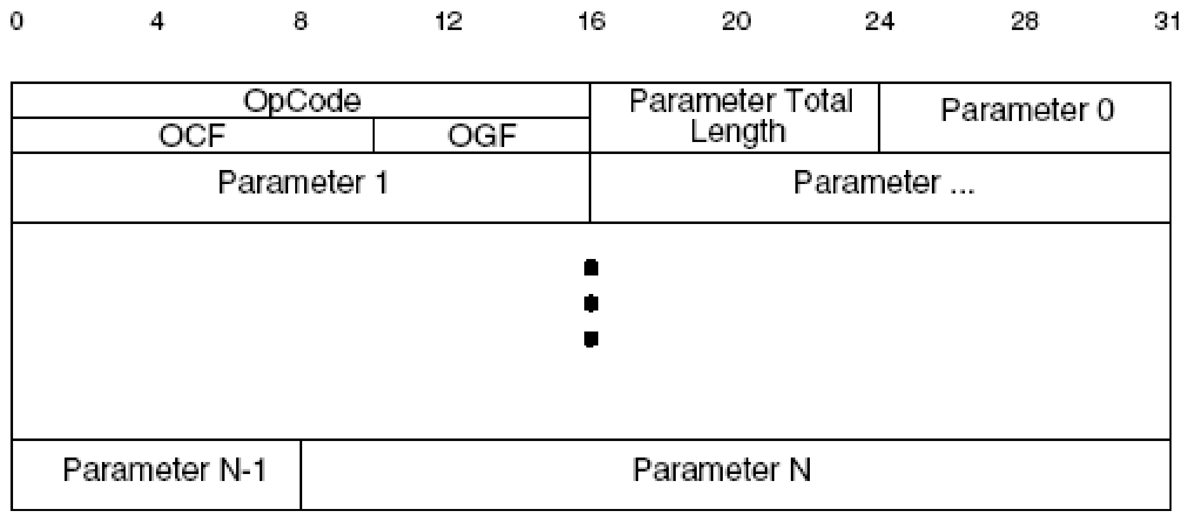


Figure 2-13: HCI command format

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 2-14. 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 2-15. 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.

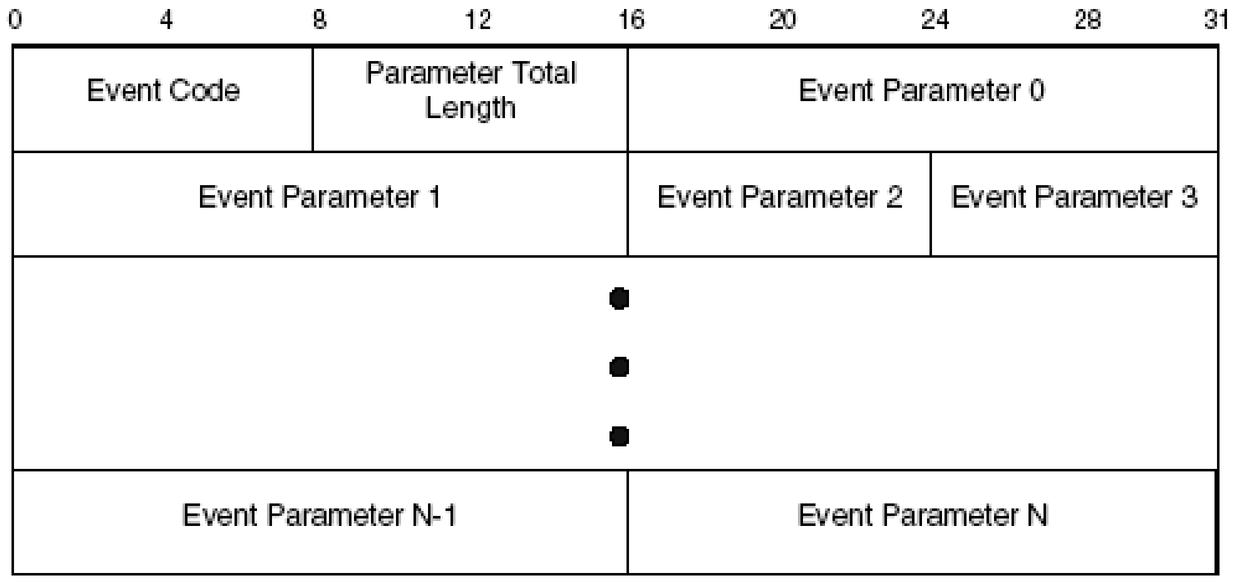


Figure 2-14: HCI event format

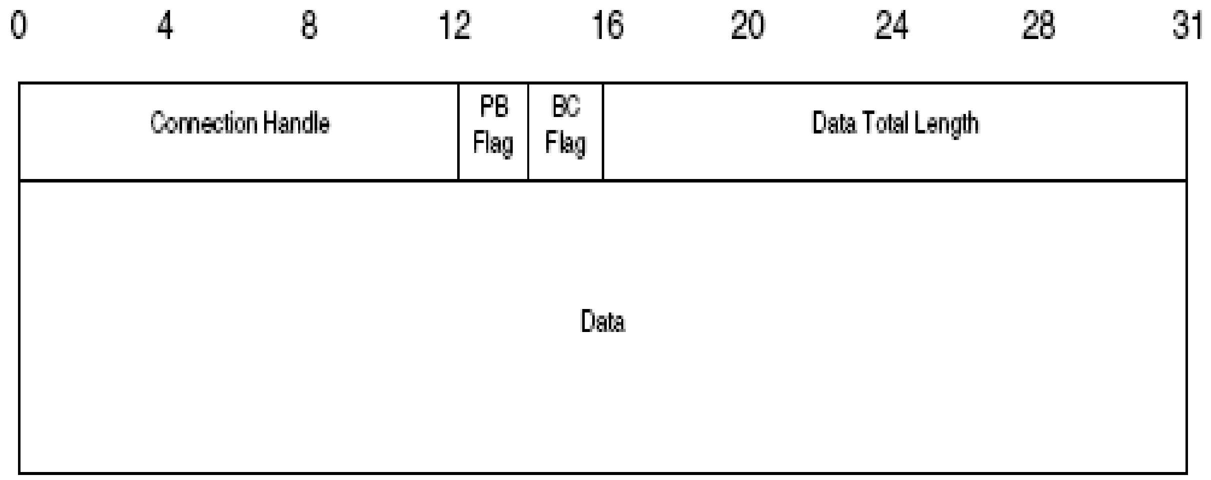


Figure 2-15: ACL data packet format

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 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 [3] (SPP) that was designed.

2.4 Bluetooth Products

There are man 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. Some stimulated interest devices will be presented next. They work under limited resources, in a hardware and software level, and provide serial connections to one or more devices.

2.4.1 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:

Manufacturer	*Bluenext Company*	*connectBlue*	*Stollmann*
Device	PROMI-MSP	SPA12i	BlueRS+E
Connections	1 to 7 or 1 to 35[12]	1 to 3	1 to 3
Data Rate	723Kbps	300-921.6Kbps	2.4-230.4Kbps

[12] Depends on the model of the PROMI-MSP

Profiles	SPP, LAP, DUN	SPP, LAP, DUN	GAP, SDP, SPP
Range	10m~100m (w/ antenna)	10m	15m
Others	4USB ports, Networking support	-	Server mode, controllable by AT commands

There is also the Teleca Comtec Bluetooth module which was used in this thesis. Further information and more thorough specifications of this module are given to the Appendix C, Chapter 9 of this thesis.

2.4.2 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.

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

In Chapter 3 an overview of the existing systems architecture will be presented. The existing systems that were developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete will be analyzed so that in the following chapters the importance of the contributions of the Bluetooth Reconfigurable Run Time environment will be thoroughly examined.

3. Existent & New Architecture

The basic motivation and incentive for this thesis was a project that started in the Microprocessor and Hardware Laboratory of the Technical University of Crete in late 2005. The project was given the name of Parrotfish which is a small fish found in Florida waters that can change gender as needed under group dynamics [6]. The basic idea of the Parrotfish project is to create an environment where a Field Programmable Gate Array (FPGA) will be (partially) reconfigured through a wireless medium (in this case Bluetooth) in a distributed ad-hoc community. The Bluetooth Reconfigurable Run Time Environment thesis' target is to transfer efficiently and wirelessly the data needed for the configuration of an FPGA in a large community of nodes. The existence of two previous theses that were focused in peer to peer communication with the use of the Bluetooth protocol was vital for the success of the present one.

The first thesis was involved with the implementation of a simple Bluetooth host controller and it was developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete by Christos Strydis (BlueApplE-BlueBridge, 2003) [7]. That thesis gave a big thrust to the continuance of the research on the Bluetooth protocol. In 2004 Politarhos Elias developed the Bluetooth Multi UART system in the Microprocessor and Hardware Laboratory of the Technical University of Crete (2003) [8]. The structure of the embedded system that had been developed in the Bluetooth Multi-UART (BluMiu) thesis was based on the initial foundation that was set by the architecture of the previous application platform that was then developed (BlueApplE-BlueBridge by Christos Strydis), but its architecture had evolved into an entirely different than that of BlueApplE-BlueBridge. Finally, it must be noted that the present thesis is a subsystem of the Parrotfish project (as it is described in chapter 4) and as far as the FPGA configuration is concerned it interacts with the thesis developed by Dionysios Efstathiou in 2002. Efstathiou's diploma thesis was developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete in

2002 (Design and Implementation of a Vendor-Independent Universal Programmer for FPGA Technology, [9]). In the following sections the architecture of the previous theses (BlueApplE-BlueBridge and Bluetooth Multi-UART) will be described, their disadvantages will be shown (in order to understand the mere contribution of the BluReRun), and the architecture of Bluetooth Reconfigurable Run Time (BluReRun) environment will be described in general. Also, in section 3.3 some general characteristics of the Efstathiou's thesis [9] will be given so that the BluReRun's architecture will be better understood. BluReRun's architecture as well as validation and implementations will be specified in detail in chapter 4 and 5.

3.1 BlueApplE-BlueBridge system

BlueApplE-BlueBridge is an applications environment, developed at the Microprocessor and Hardware Laboratory (MHL), as part of the diploma thesis by Christos Strydis [7]. The most important component of this system is a microcontroller (HOST) that is used to control all the other components that BlueApplE-BlueBridge consists of.

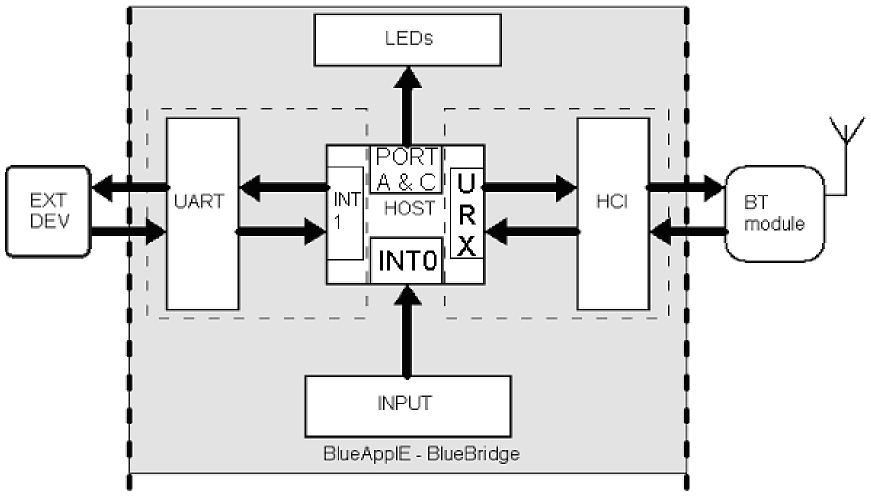


Figure 3-1 BlueApplE – BlueBridge

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 3-1). Through the BlueApplE-BluBridge description its advantages and disadvantages will be stressed out.

3.1.1 Software of BlueApplE - BlueBridge

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.

3.1.1.1 BlueApplE software architecture

A generalized flow chart of Blue Apple's software architecture and operation can be seen in Figure 3-2 and an explanation of its functions follows.

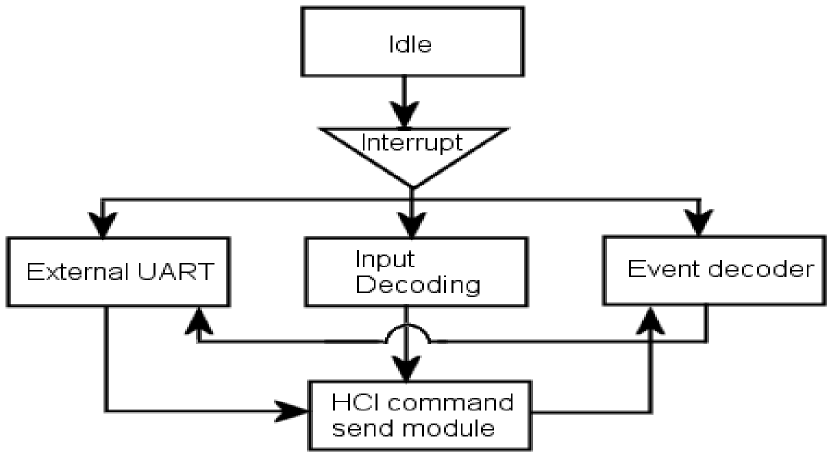


Figure 3-2. BlueApplE operation flow chart

The idle process puts the μC in a power saving mode until an interrupt is triggered. When the software detects a supported interrupt, it leads the μC to execute the appropriate module. After the end of each of the modules' jobs, the μC 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 3-2. 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 μC'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 μC 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 3-2) 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 μC's internal UART to transmit the bytes read. After commands are executed by the BT module, a Bluetooth event is usually returned to the μC. 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.

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

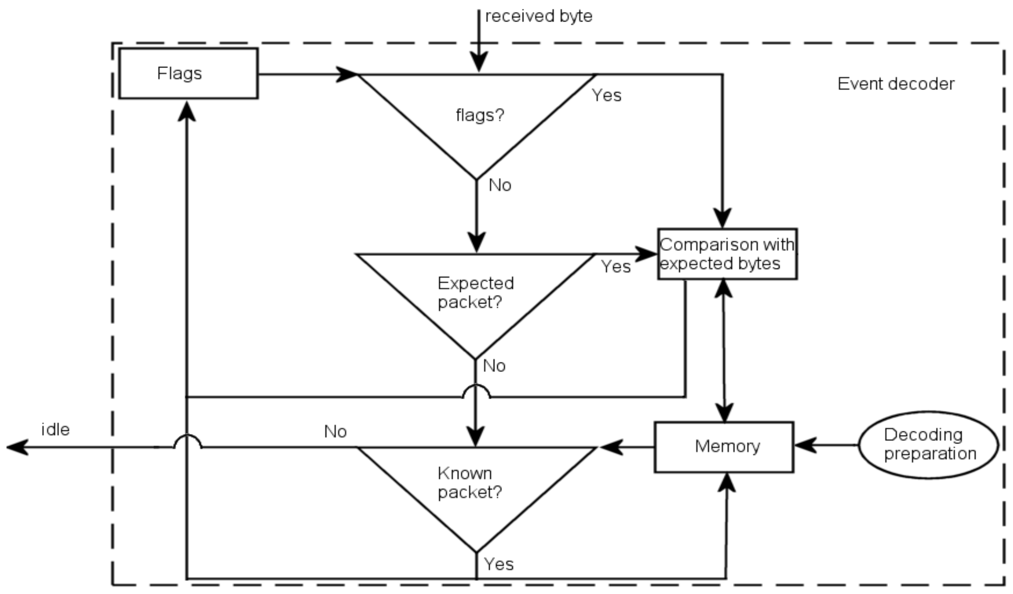


Figure 3-3 BlueApplE event decoding module

Bluetooth commands implemented in BlueApplE are stored in the μC's memory in two copies: one in the RAM of the μC and one in the non-volatile flash memory (FLASH) it has for storing the program code. When the μC 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 μC used has *only* 512Bytes of memory.

As a conclusion, the BlueApplE 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 BlueApplE
- The decoding preparation module would be useless, if a better event decoding module was implemented

3.1.1.2 BlueBridge software architecture

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

For the BlueBridge to work both ends of two connected BlueApplE devices must start the BlueBridge application. The handshaking process used is depicted in Figure 3-4. 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 BlueApplE 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*. [1] *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. 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.

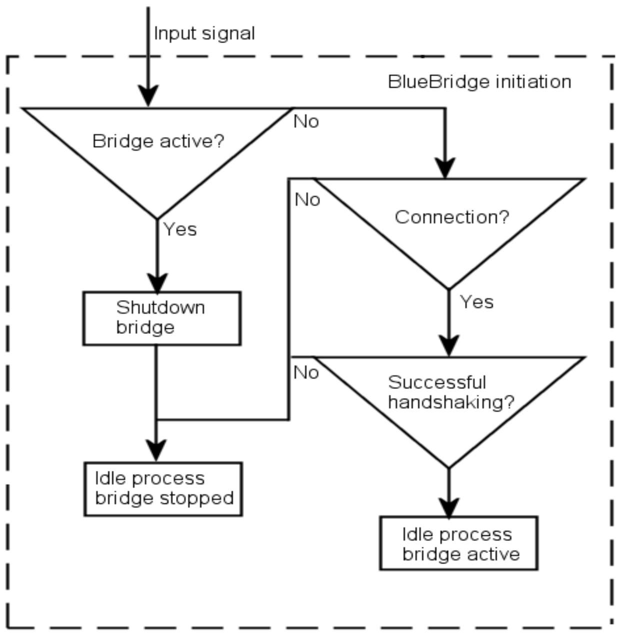


Figure 3-4 BlueBridge Initiation

The decoding of data packets is done using the same mechanism that BlueApplE uses to decode events. (Figure 3-5). 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[13] 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.

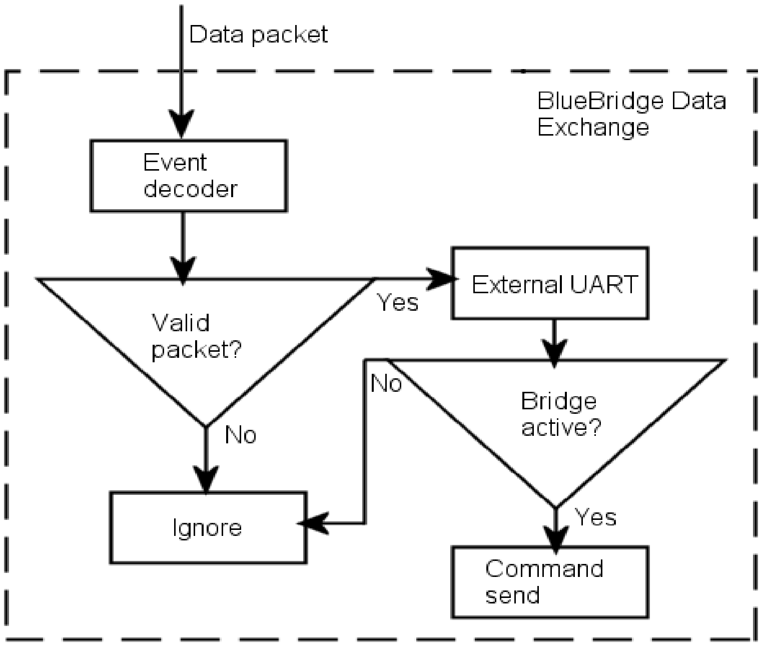


Figure 3-5. BlueBridge data exchange through the BlueApplE

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[14] 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

[13] 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

[14] The default data rate of the BT module

system's architecture and functionality, the BlueBridge application also has certain disadvantages, in addition to those of BlueApplE:

- 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

So in general the basic disadvantages of the BlueApplE – BlueBridge system were:

- 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 BlueApplE
- The decoding preparation module would be useless, if a better event decoding module was implemented
- 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

3.2 Bluetooth Multi-UART system

Bluetooth Multi-UART (BluMiu), developed at the Microprocessor and Hardware Laboratory and designed by Elias Politarhos in 2004 [8], 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 μC that was used, so that it could cover the requirements. The AT80S8515 was changed by the ATmega161 AVR.
- 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-point connections could be achieved.

BlueApplE 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, μC 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 BlueApplE 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 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.

1. The differentiated decoding of incoming events. BlueApplE 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 μC 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.

2. 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 μC 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 μC is executing an ISR and a subsequent interrupt is triggered, data vital for the ISR running is stored in the μC's RAM, so that the μC can execute the ISR for the new interrupt and when it ends the first ISR will be resumed. This fact allows the μC to execute all the interrupts that occur, internal and external, without losing any data or missing any interrupts and, thus, decreasing the Host's (AVR) reliability to the external device.

3.2.1 The BluMiu software Architecture

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 in real time. The software architecture of the client and the server is shown in Figure 3-6.

The idle process makes the μC to enter a power saving mode until an interrupt is triggered. When that happens, the μC 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.

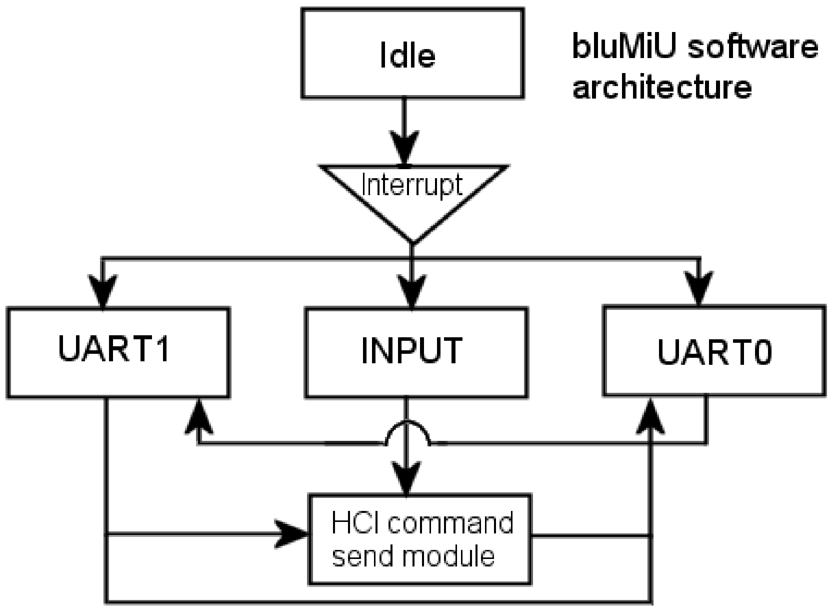


Figure 3-6 BluMiu software architecture

The UART0 as well as the UART1 were the most important modules of the BluMiu architecture. In the BluMiu the Host which is the AVR supports the differentiated decoding of packets. However, this is not done efficiently. For example in order to send data through one Bluetooth module to another it is essential to add a large number of header bytes to the actual data packet. This has been eliminated in the BluReRun, by creating optimized UART decoder modules; thus improving the whole system's performance. This will be thoroughly examined in the next chapter.

However, the Bluetooth Multi UART environment had some basic disadvantages that were important and were resolved in this thesis:

- The format of the packets accepted from and transmitted to the Bluetooth controller and external device had to be changed, as far as the interaction between the BluMiu and the top levels of the architecture is concerned, in order to support efficient FPGA re/configuration even in point to point connections.
- No true point to multipoint environment was supported by the BluMiu due to hardware limitations
- The merging of a master and a slave in one host was not supported.
- No addressing protocol was implemented
- The Bluetooth module used in previous theses was specified by the Bluetooth spec 1.0b therefore it was not up to date with current specifications (Bluetooth spec 2.0)
- No dynamic initialization and configuration of commands were supported. In BluMiu everything must be done by a terminal window or through the button input

3.3 The Hardware Programmer and Tester (HPT)

The purpose of the thesis that was developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete by Dionysios Efstathiou in 2002 [9], was to develop of a vendor – independent universal programmer for FPGAs (this programmer was named HPT). The Hardware Programmer and Tester (HPT) is the core of the thesis. It is a space-efficient, pre-engineered high-density configuration solution for programming testing and upgrading FPGA based systems. It can offer adequate solution to the above problems as an inexpensive and generic programmer. The HPT can be utilized for system prototyping and testing, or as intelligent host responsible for configuring multi-FPGA systems. The architecture of the HPT will be described in general in section 3.3.1 since it is vital to understand the way it works due to its interaction with the present thesis. Finally a general comment on FPGAs will be given in section 3.3.2 since one of this thesis' purposes is the programming of an FPGA through the interaction with the HPT.

3.3.1 HPT Architecture in general

The HPT has an RS232 link to the PC and an on-board flash memory both controlled by an intelligent host (either a microprocessor or an FPGA). The Run-Time Environment in the PC is responsible for downloading the appropriate CDF file (configuration bit-streams along with the necessary HPT instructions) to the HPT board either for immediate usage or for long-term storage in the flash memory. A detailed diagram of the HPT is shown in Figure 3-7, where configuration data and HPT instruction are transferred through the RS 232 interface to the HPT core module. The Run Time Environment is responsible for the data transfer to the HPT. Upon reset, the HPT system is in an idle mode. Depending on the incoming instructions the HPT core module decides whether to proceed in data store, in the flash storage media, or to operate directly through the RS 232 interface and configure (or test) the connected FPGA(s) based system. The instructions can be divided in "protocol" instructions and

“Programming” instructions. For example, the command that stores data in the Flash media is categorized as a protocol instruction. The microcontroller unit (Figure 3-8) of the HPT core “parses” the incoming instructions and acts accordingly by issuing commands to the various HPT interfaces. The fact that the timing requirements of some FPGAs are demanding and the whole configuration process may be complicated makes the need for branching instructions mandatory.

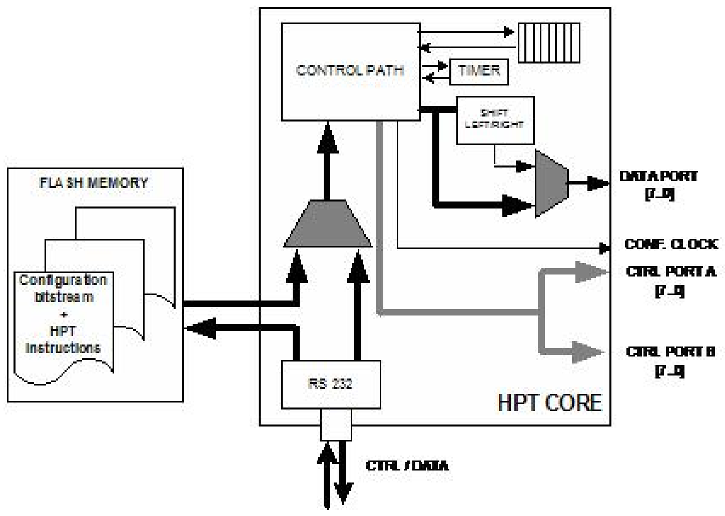


Figure 3-7: The HPT

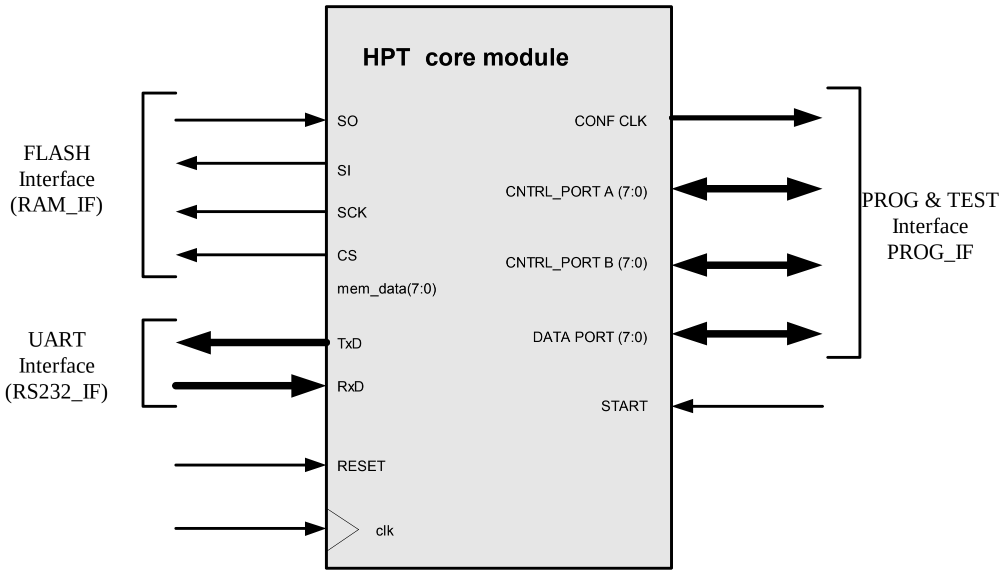


Figure 3-8. The HPT microcontroller

Consequently, the HPT core module must be equipped with a cache memory in order to store portions of instructions for branching. The reason for implementing a caching policy in a memory block different from the flash storage media is the need for accelerated time response and instruction-exclusive storage media.

3.3.2 FPGA history and architecture in general

There are two types of programmable ASICs including programmable logic devices (PLDs) and field-programmable gate arrays (FPGAs). This text will be focused exclusively on FPGAs.

In the mid 1980s a new technology for implementing digital logic was introduced, the field-programmable gate array (FPGA) [5]. These devices could either be viewed as small, slow mask programmable gate arrays (MPGAs) or large, expensive programmable logic devices (PLDs). FPGAs were capable of implementing significantly more logic than PLDs, especially because they could implement multi-level logic, while most PLDs were optimized for two-level logic. Although they did not have the capacity of MPGAs, they also did not have to be custom fabricated, greatly lowering the costs for low-volume parts, and avoiding long fabrication delays. While many of the FPGAs were configured by static RAM cells in arrays, (SRAM), this was at first viewed as a liability by potential customers who worried over the chip's volatility. Antifuse-based FPGAs were also developed, and for many applications were much more attractive, both because they tended to be smaller and faster due to less programming overhead and also because there was no volatility to the configuration. The major disadvantage to Antifuse technology FPGAs is that once they are programmed the process cannot be reversed. They are not re-programmable.

In the early 1990s there was a growing realization that the volatility of SRAM-based FPGAs was not a liability, but was in fact the key to many new types of applications. Since a completely electrical process could change the programming of such an FPGA, much as a standard processor can be configured to run many programs, SRAM-based FPGAs have become the workhorse of many new re-programmable applications. An IC foundry produces FPGAs with some connections missing. The user can perform design entry and simulation. Next, special software creates a string of bits describing the extra connections required to make the design—the configuration file. There is no customization of any mask level for an FPGA, allowing the FPGA to be manufactured as a standard part in high volume.

FPGAs are popular with Microsystems designers because they fill a gap between TTL and PLD design and modern, complex, and often expensive ASICs. FPGAs are ideal for prototyping systems or for low-volume production. FPGA vendors do not need an IC fabrication facility to produce the chips; instead, they contract IC foundries to produce their parts. All FPGAs have certain key elements in common. All FPGAs have a regular array of basic logic cells that are configured using a programming technology [10]. The chip inputs and outputs use special I/O logic cells that are different from the basic logic cells. A programmable interconnect scheme forms the wiring between the two types of logic cells. Finally, the designer uses custom software, tailored to each programming technology, and FPGA architecture, to design and implement the programmable connections. The programming technology in an FPGA determines the type of basic logic cell and the interconnect scheme. The logic cells and interconnection scheme, in turn, determine the design of the input and output circuits as well as the programming scheme. The programming technology may or may not be permanent. The permanent programming in one-time programmable (OTP) FPGAs cannot be undone. Re-programmable or erasable devices may be reused many times.

3.4 The New System: BlueReRun

As it was noted in the introduction of this chapter the Bluetooth Reconfigurable Run Time Environment (BlueReRun) which was developed in the Microprocessor and Hardware Laboratory of the Technical University of Crete and designed in this thesis is part of a larger project. The Parrotfish's (the project name) target is to create a run time environment where a Field Programmable Gate Array (FPGA) will be (partially) reconfigured through a wireless medium (in this case Bluetooth) in a distributed ad-hoc community.

The basic targets of the BlueReRun are the following:

- To improve the performance of the BluMiu system,
- To create a system that will interact correctly with the Hardware Programmer and Tester so that efficient FPGA re/configuration would be guaranteed,
- To create an addressing protocol that would forward correctly and efficiently the data required for the re/configuration of an FPGA, in an ad-hoc community
- To merge in one device the master-slave characteristics so that every node in an ad-hoc community (piconet in Bluetooth) would be able to change from master to slave when requested by the Hardware Programmer and Tester or by other factors (such as an external device other than the HPT)
- To create at the same time a system that can also stand alone apart from the interaction with the HPT. This system must support any data or command exchange within an ad-hoc community of nodes in an efficient and errorless way.

In order to understand the importance of the contributions of the BluReRun it is necessary to understand the mode of communication between the different levels of the Parrotfish project. A Parrotfish community is a community that has a number of nodes which have the format shown in Figure 3-9. In this figure 3 levels can be clearly noted. The first one is the Bluetooth layer (layer 1) followed by the HPT layer (layer 2) and finally the top level layer is the FPGA Layer (layer 3). All the nodes noted in the figure, have this format. The BluReRun's target is to create an efficient *layer 1* so that the data (meaning the .bit file that the FPGA requires for the re/configuration) would be ideally and efficiently forwarded to different nodes.

The first layer must interact correctly with the second layer, which is the leader of the Parrotfish system. The BluReRun is capable of making decisions only as far as the way that the data could be distributed among the different nodes. The target of the data (where the data would be sent) is determined only by the HPT. Also, an HPT and BluReRun command interaction must be supported.

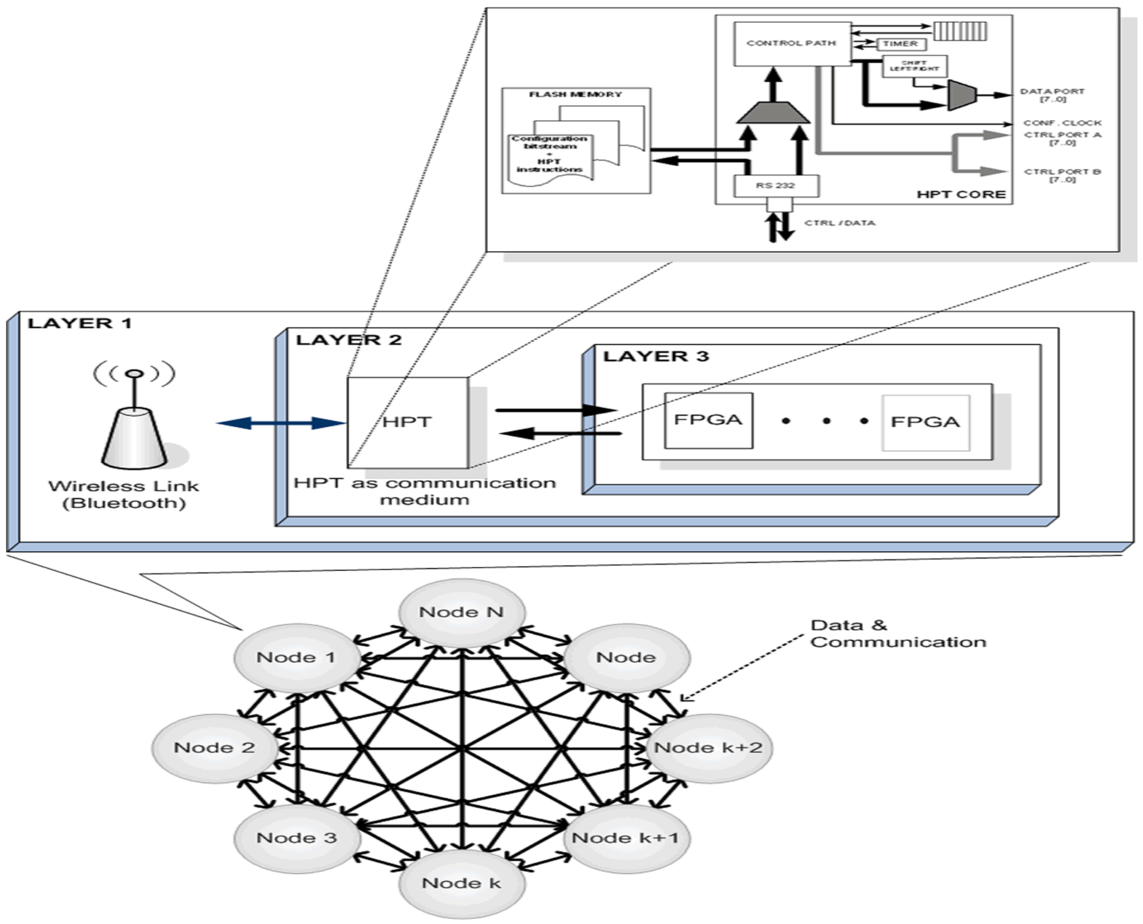


Figure 3-9: The Parrotfish community and the different layers

The BluReRun's software and hardware specifications must be the same in all the nodes that form the ad hoc community. The BluReRun must overcome the disadvantages created by the previous systems as well.

At this section a general and precise description of the BluReRun system was presented. In Chapter 4, a thorough and detailed study of the system as well as its parts will be given in order to understand its contributions to FPGA programming and wireless data configuration.

4. BluReRun Architecture

In chapters 2 and 3, an overview of the lower levels of the Bluetooth stack and the existing Bluetooth enabled system developed in MHL in the context of Christos Strydis' thesis (BlueApplE-BlueBridge) [7] as well as the research that Elias Politarhos did for his own diploma thesis (BluMiu) [8] were presented. We explored some of the disadvantages that the BlueApplE-BlueBridge and BluMiu systems possessed and saw what needed to be improved. The new system (Bluetooth Reconfigurable Run Time Environment) reduces and in some cases it even eliminates the flaws of the existing systems. At the same time, the BluReRun system introduces an integral system of re/configuring FPGAs in a wireless ad hoc community.

BluReRun offers the following characteristics that make it better than the existing one and at the same time ideal for FPGA wireless re/configuration:

1. *Piconet-based master-slave architecture*. In BluReRun we have three (3) platforms (three because we have only three point to multipoint BT modules) where the host (AVR) communicates with the Bluetooth module through USART 0 and with the HPT through the USART 1. Effectively, through the USART 1 and through the HPT the data can be sent to an FPGA. As stated in chapter 3 and in the introductory chapter of this thesis, the BluReRun's major target is to be able to send the data (bit stream file) efficiently to an FPGA (through the HPT) in a Bluetooth ad hoc community which is comprised of three nodes. Therefore, the BluReRun must act as an efficient data transporter between different nodes of an ad hoc network.
2. *Point to multipoint connections* supported and most importantly implemented.
3. *Addressing protocol implemented* since we have more than two Bluetooth devices in the vicinity area. Also the data must be forwarded correctly

according to the rules stressed by the master of the piconet and the ones imposed by the Bluetooth specification version 2.0.

4. *New Bluetooth Modules have been used*. The new Bluetooth modules that have been acquired support point to multipoint connections according to Bluetooth specification version 1.2 and 2.0 (Appendix C).
5. *Dynamic HCI command configuration is supported*.
6. *Error handling for efficient data transport implemented*.
7. *A better AVR microcontroller has been used*. The new microcontroller which has new Universal Receiver/Transmitters is the ATMEGA162. It has two UARTs which were updated by ATMEL in USARTs (in Appendix B as well as in the section named BlueReRun hardware). They are called Universal Synchronous/Asynchronous Receiver/Transmitters instead of Universal Asynchronous Receiver/Transmitters. Specific information on the use of USARTs is given in Appendix B.
8. One of the trickiest parts and contributions of the BluReRun system is *the merging of master and slave in one device*. Every device (node in the ad hoc network) can be used as a master or a slave through the usage of certain commands issued by the leader of the Parrotfish project which is the HPT or by another external device.
9. *The upgrade of commands* implemented in either the master or the slave of the ad hoc network, from Bluetooth version 1.2 to Bluetooth version 2.0 has been achieved.
10. The 57600kbps transmission rate has been secured. Also the 115200 and 460 kbps transmission rates have been used.
11. *software-interrupt based multiprogramming*
12. *No push-buttons have been used*. Everything in a BluReRun node is done in a dynamic manner either with or without the use of the PC.
13. the host can be fully controlled from the device connected to the platform
14. the device connected to the platform can give commands to the BT module
15. improved data and command sending mechanism to the BT module,

16. improved decoding of incoming event and data packets from the BT module mechanism
17. improved mechanism of data exchange between a device connected to the platform and the platform itself
18. the platform reports the connected device of the addresses, the handles and the user friendly names of connected and inquired devices
19. the platform notifies the connected device whenever a disconnection occurs
20. *Connectionless data transport*
21. *Payload data packets are big enough to support the data send to the FPGA.*
22. *Multiprocessing* of the commands/events that are coming from the upper layer of the Parrot Fish system (HPT) and from the Bluetooth module.
23. *Less cost for the implementation of the BluReRun design than BluMiu.*

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

- The whole software design that contains the upper layers of the BlueTooth protocol stack and the data exchange protocol
- The hardware that is used to control the Bluetooth module, communicate with it and provide Bluetooth connection to the external device which can be either the terminal window or the HPT

In the thesis' context, three BluReRun platforms have been implemented; each one can be dynamically serve as the master or the slave regarding the status of the upper Parrotfish protocols. Firstly, the BluReRun data transfer protocol will be examined. Then, all the contributions of this thesis will be presented and analyzed.

4.1 BluReRun data transfer protocol

It is important for an ad hoc community to have efficient and errorless data transport between each node. The exact protocol, which enables BluReRun 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 2.0, only one connection at a time is dealt within this profile [3]. Hence worth, only point-to-point configurations are considered at a time (even at point to multipoint connections). This means that (as seen in section 2.3.2.4) in each timing slot, only two Bluetooth devices can exchange data (master in even time slots and slave in odd slots).

4.1.2. The BlueReRun protocol approach

The BlueReRun protocol had, from the beginning of this thesis, to be efficient enough and very meticulously specified in order to support not even point to point but also point to multipoint errorless packet exchange. Before examining the BlueReRun protocol approach it is essential to understand the structure that a node in the Parrotfish project has.

4.1.2.1 Architecture of the Parrotfish Node

As described, in chapter 3 the Parrotfish Project's targets is to efficiently re/configure any FPGA device wirelessly in an ad-hoc environment. The Parrotfish is, therefore, comprised of many nodes which have the same structure. The structure of a node is depicted in figure 4-1.

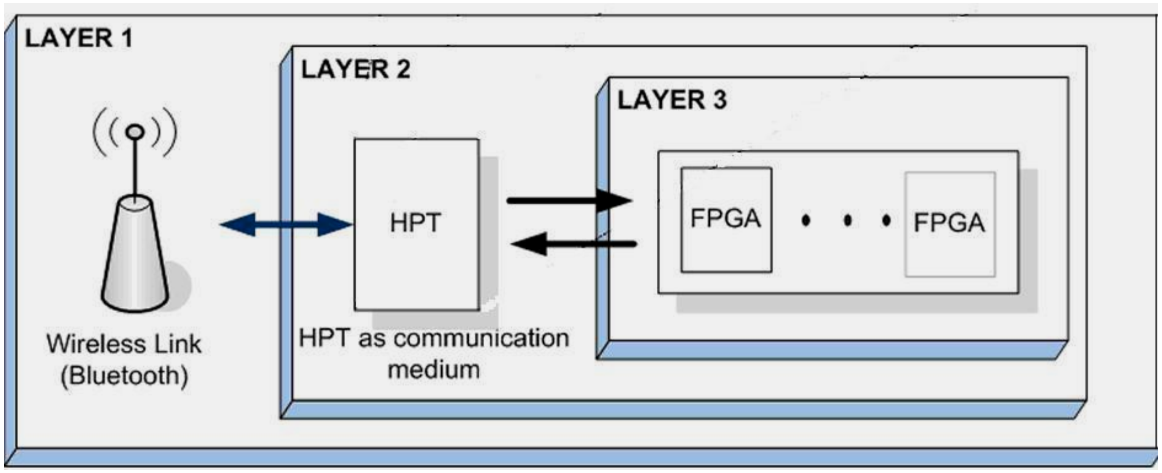


Figure 4-1: The Parrotfish node

The Parrotfish node is partitioned into three inner-communication layers. The "lower" layer (layer1) is the physical and data-link layer (this is the layer where the BluReRun thesis is focused). It consists of the Bluetooth module and its Host. The "middle" layer (layer2) is the control medium and the "upper" layer (layer3) is the reconfigurable section of the sensor node. The "middle" layer acts as the intermediate between the reconfigurable layer and the rest of the network. It consists of the HPT (Hardware Programmer & Tester), an 8-bit microcontroller and a memory storage unit [9]. All incoming data from the network that reach the node are collected by layer1. After they have been "stripped" from any Bluetooth related data, they are forwarded to layer2. In turn, the middle layer "parses" the received data and according to their use, they are forwarded to the reconfigurable section (layer3) or kept as needed. Vice-versa, layer2 collects processed data and requests from layer3 and either redirects them to the upper layer for transmittance to the network or processes them. It also has the ability to dynamically (re)program or read back the configuration bit stream of the node's reconfigurable section (layer3). The architecture of the Parrotfish node offers a level of transparency between the different layers. Each layer only communicates with its adjacent layer. The Bluetooth layer (layer1) interfaces only with the control medium (layer2), inside the node. The existence of the reconfigurable section, its functionality and even its presence, is completely hidden from layer1. The communication layer, on the other hand, communicates with both layers.

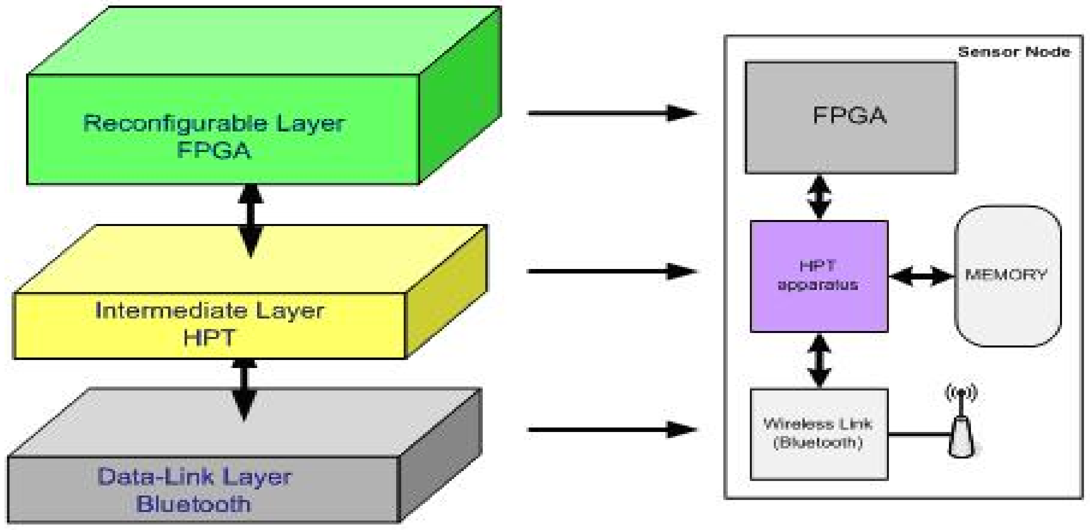


Figure 4-2: Layers of transparency

The “middle” layer communicates with the “lower” layer (layer1: Bluetooth) in a fixed serial manner. It forwards “raw” data to layer1 (in the form of HPT packets), which in turn converts them in the Bluetooth compatible packets, and transmits them to the network. In reverse, layer1 receives ACL data packets from the network and after they have been converted to HPT packets, they are sent, serially, to the middle layer. Thus the hardware and software of the wireless link layer is completely transparent to the medium layer [12]. The above communication restrictions apply also to the reconfigurable layer of the node. The “line of sight” (“outside world”) of the 3rd layer is the “middle” layer (layer 2). It receives requests only from it and responds accordingly. The third layer communicates with the “middle” layer through a serial link. This is mainly due to the limited resources of the microcontroller plus the need for a unified communication link in all nodes.

In Figure 4-2 the different layers of transparency and what they are comprised of is depicted.

The ultimate purpose of the BluReRun is to create an efficient and low cost Data Link Layer in the Parrotfish node. This layer must function as the transporter of the data required by the HPT (Intermediate Layer) to re/configure dynamically an FPGA. At the same time, as stated in the beginning of this chapter, the BluReRun must also act “alone” in an ad hoc environment.

4.1.2.2 The BluReRun protocol in the Parrotfish project

The BluReRun protocol is very methodically specified in order to secure efficient and errorless data/command transport in a point to multipoint environment such as the environment that the Parrotfish project demands. It needs the external devices connected to the BluReRun (either the HPT or any other device such as a PC) to know that special packets can only be received by the Host (the AVR ATMEGA162 that controls all the functions of the data link layer as it will be studied in section 2.3). These packets are called HPT packets (the name given by the HPT who is the control medium of the Parrotfish project). The format of the HPT packets is shown in Figure 4-3.

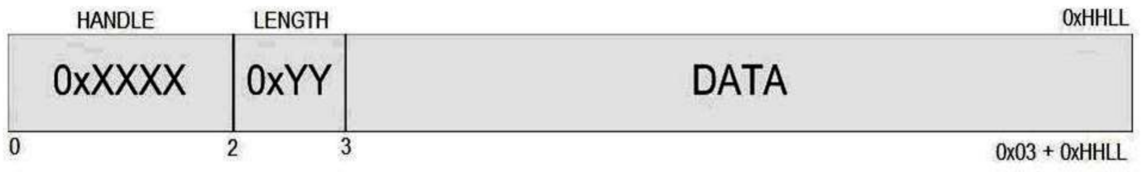


Figure 4-3: HPT packet format

In the above format the following fields can be noted:

a) *Handle [2 bytes]*: This is the ID of the target device. It is 2 bytes long and all the handles of the targeted devices are reported through the Host to the external device or the HPT. The Handle is two bytes long since that is required the Bluetooth specification profile v2.0 [4]. The Handle is returned to the Bluetooth Host by a connection complete event. The form of the handle is created in such a way that supports point to multipoint connections.

b) *Length*: This is the total Length of the Data of the HPT packet. It complies fully with the data payload that the HPT (which is the control medium of the Parrotfish project) sends. It is 1 byte long since the HPT sends data of no more 256 bytes to the FPGA (layer 3 in Figure 4-1).

c) *Data*: This is the actual data that one HPT sends to another HPT in a node of the ad hoc system, through the BluReRun. The BluReRun is not interested at all in the contents of this data. It is interested only in the efficient transport of ALL the contents of the data in the ad hoc system.

The handle format is in big endian order, since the HPT sends the HPT packets to the BluReRun Host in such an order. However, the Bluetooth firmware supports only bytes in little endian order. Therefore, in the software decoding (as it will be studied in the Software Architecture section) the essential transformations are made in order to ensure little endian transfer of data.

When such an HPT packet is received from the host (AVR ATMEGA162) the validity of the handle will be checked by the BT module. If the handle is valid, a decoder will transcribe the packet received to an ACL data packet (its format is depicted in 4-4, 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 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 BluReRun the CID 0x0040 is used, which is not reserved for any BlueTooth profile.

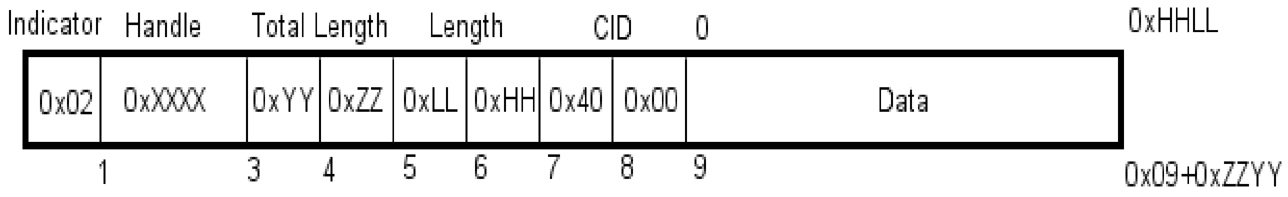


Figure 4-4 BT module acceptable ACL data packet used by BluReRun

Every number used in the header is in a little endian byte order. After the BluReRun 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. The above procedure will occur every time that a data exchange between two or more BluReRun platforms shall happen either in point to point or point to multipoint environments.

As explained in the contributions of the BluReRun thesis, each Host (and therefore each node of the ad hoc network) can function as a master or a slave. More about this BluReRun's characteristic will be explained in section 4.3. When a master/slave of the BluReRun platform, transmits an ACL data packet to another master/slave of the ad-hoc network, then the BT module receives the packet and forwards it to the host. Finally each node shall transcribe the ACL data packet received, to a HPT data packet (Figure 4-3) and transmit it to the HPT or another external device. Certain receive and transmit control modules were implemented. These modules shall be explained in section 4.3.

The following sections describe in detail the core of the BluReRun's system hardware and software architectures.

4.2 BluReRun Hardware

The BluReRun hardware architecture was one of the most difficult sections of the BluReRun thesis. From the beginning of this thesis, the hardware had to coincide with the software architecture in order to guarantee an efficient transporter of data and commands in an ad hoc network. In the following sections the hardware components of BluReRun will be described.

4.2.1 HOST

The most fundamental component of the BlueReRun system is the HOST. The Host, which is an ATMEGA162 AVR from ATMEL [11], offers the necessary processing power to execute the upper Bluetooth protocol layers as well as the essential software and hardware processing for the efficient data transport between many nodes of an ad hoc system. In Figure 4-5 the HPT hardware architecture is depicted. In this thesis three BluReRun nodes with three Hosts were implemented. The analytical schematic and PCB design of each one of those nodes is given in Appendix A.

Each BluReRun node in the Parrotfish ad hoc network has one Host. Each host has the Bluetooth master and slave characteristics embedded in its architecture. All the hosts have the same characteristics and the same capabilities.

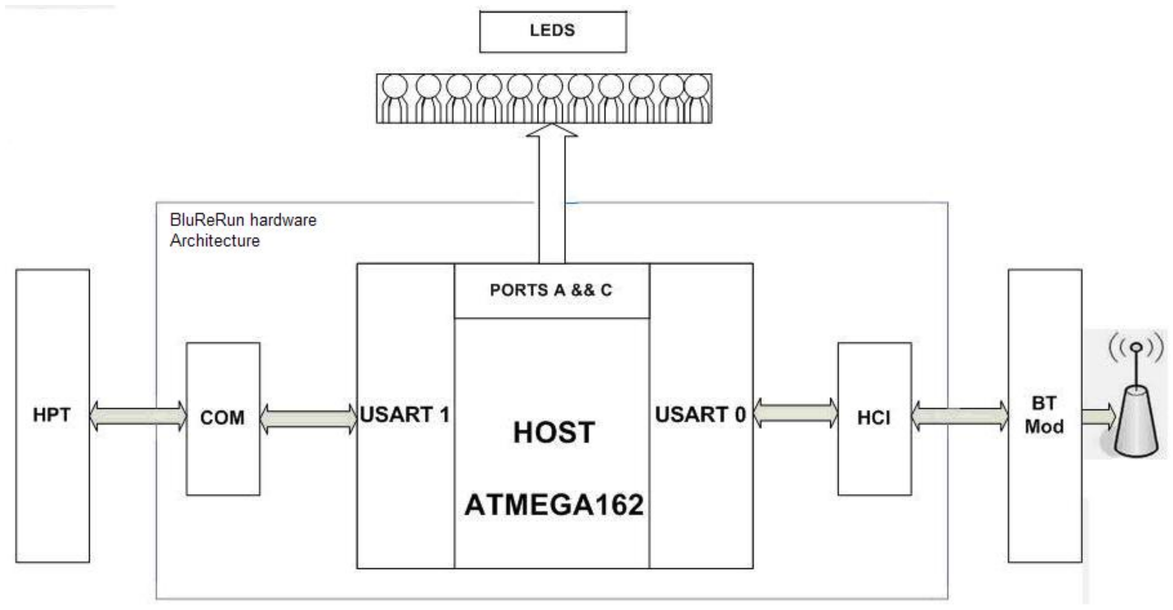


Figure 4-5: BlueReRun Hardware Architecture

The Host must successfully accomplish, every time it is asked, the following tasks:

a) to communicate correctly with the BT module via USART 0
b) to communicate efficiently with the HPT and any other external device via USART 1
c) to execute different commands issued by the HPT or by the host itself in a dynamic way
d) to secure the safe migration from master to slave or vice versa when requested by the HPT or by another external device
e) to discover Bluetooth enabled devices in its vicinity dynamically or when requested by the HPT
f) to connect to them or disconnect from them in a dynamic way
g) to exchange data or command packets with any one of them guaranteeing the efficient data transport between the other nodes of the ad hoc network,
h) to manage the data flow between the BT module and the HPT,
i) to correctly decode the incoming command and/or data packets, either they are being received from the BT module or the HPT,

j) to transform the HPT packets to ACL data packets, compatible with the Bluetooth protocol, and vice versa, and

k) to present in the board leds the correct decoding of packets and events

The Host of any BluReRun node, since it is the core of the system, should be powerful enough to support multiprogramming modes and also to guarantee:

- ✓ Availability
- ✓ Reliability
- ✓ Low cost
- ✓ Pin compatible with ATMEGA161
- ✓ Fulfillment of the hardware requirements set by the BluReRun design

Microcontroller	Needed	AT90S8515	ATmega161	ATmega162
SRAM	*900bytes*[15]	✗ 512bytes	✓ 1024bytes	✓ 1024bytes
UARTs or USARTs	*2*	✗ 1	✓ 2	✓ 2 USART
Program memory	*12Kbytes*	✗ 8Kbytes	✓ 16Kbytes	✓ 16Kbytes
Multiplier	*1*	✗ 0	✓ 1	✓ 1
I/O lines	*28*	✓ 32	✓ 35	✓ 35
Registers	*32*	✓ 32	✓ 32	✓ 32
8 - 16 bit timers	*2*	✓ 2	✓ 3	✓ 5

Figure 4-6: The Host Resources

The upgrade of the Host from ATMEGA161 (which was used in BluMiu [13]) to ATMEGA162 was fulfilled. The basic reason was the hardware and software capabilities that the ATMEGA162 could offer on the grounds of Timer/Counters

[15] At least 2x256bytes for USART Transmit buffers, 10x10bytes for inquiries, 7x8bytes for connections, 10x10 bytes for connection/disconnection purposes

and USARTs. The ATMEGA162 offers up to 5 different Timer/Counter modules for 8 and 16 bit timing modes. In the BluReRun design only 2 are used for error handling purposes. Also, as far as the interaction with the HPT (the external device) and the BT module is concerned, the ATMEGA162 is using the USARTs. In the table 4-6 all the necessary resources for the BluReRun design and all the different hosts implemented in all the previous designs (BluAppiE and BluMiu) is depicted in Figure 4-6.

As it can be seen from the above figure the ATMEGA161 and the ATMEGA162 fulfill the necessary resources for the function of the BluReRun design. However, the usage of USARTs was the one reason that led to the migration from ATMEGA161 to ATMEGA162. The USART stands for Universal Synchronous/Asynchronous serial Receiver and Transmitter. The USARTs are embedded in the microcontroller used (ATMEGA162 [11]). The basic difference between the UARTs and the USARTs is the use of two different receive buffers corresponding to USART0 and USART1. This function was important for the BluReRun design since the possibility of having a packet missed was decreased; thereby, the efficiency of the whole system was increased. Also, the USART with the use of a flag in UCSRA Register (UMSEL) can have different modes of clock generation. Another USART-UART difference is that the first one can support synchronous or asynchronous operation. Further information on USART is given in chapter 8 (Appendix B).

4.2.2 BT module

The other essential component of the BluReRun system couldn't be other than the one that provides the system with Bluetooth connectivity, which is the BT module. The BT modules were a kind donation of Teleca Comtec AB in 2006. They support point to multipoint protocols and are described in detail in Appendix C. As seen in the chapter 2, they embed the BlueTooth Radio, the Baseband, the Link Manager, a UART and a USB HCI [14] [15]. This means that the BT modules used accept Bluetooth commands, execute them and return their outcome. All the connectivity between the Bluetooth modules and the Host is embedded in the Host core and done through the USART 0 which is the Host Controller Interface.

4.2.3 HCI & COM

The Host makes use of the HCI and COM interfaces to communicate with the BT module and the HPT (or any other external device). The internal USARTs of the ATmega162 are used to communicate with both of them. A default baud rate of 57.6Kbps is used and 256byte buffers are associated with the Host's USARTs, implemented in software and supported by the SRAM memory. Each BluReRun hosts supports also Baud Rate of up to 460.8 Kbps but such high speeds are not used since there is no efficient error algorithm that can support efficient data transmission over those Baud rates.

4.2.4 LEDs

Two groups of Leds have been created in the BluReRun design. The one group is used for decoding and debugging purposes, in a binary scale, while the other is used for command and status messages. The importance of the LEDs in the whole system is crucial since if every BluReRun node is considered as a black box then the user of the system can understand or debug the system by looking at the diodes.

4.2.5 EXT DEV

The efficient data transmission/reception through the HPT packets, between the HOST and the HPT (or any other external device) is a very important task of the BlueReRun node. 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 BluReRun data transfer protocol so that it can exchange data with other devices that support the BluReRun data transfer protocol. This connection is achieved through the COM interface. The COM interface is the USART 1 module of the

Host. Especially, on the boundaries of the Parrotfish Project each external device (which is the HPT, layer 2, Figure 4-1) must be able to communicate with any other HPT of the ad hoc network (through the BluReRun system, thereby securing transparency) in order to re/configure any FPGA.

4.2.6 BluReRun hardware cost

The following hardware components were used for the development of each one of the BluReRun nodes:

Parts	**Cost**
ATMEL ATMEGA162	*$5*
TELECA COMTEC BT Module	*$30*
ADM202EAN RS-232	*$1.5*
Various components (leds etc)	*$2*
PCB Construction	*$10*
Total Cost	**$48.5 ~ $50 (about 44 euros)**

Table 1: Cost of BlueReRun design

By using these components the cost of the platform's hardware would be about 50$. In BluMiu and BluAppiE design the cost was more than 90$ for implementation. Since a low-cost BlueTooth design is a primary objective of this thesis, the cost could be further reduced by replacing the Bluetooth module by a BlueTooth RF IC and a BlueTooth Baseband controller. 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.[16] SKY73001 BlueTooth RF Transceiver. The XE1402 is a BlueTooth Baseband controller that supports only data transfers

[16] 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]

(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 $20 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 BluReRun system.

4.3 BluReRun software

The development of the software that was embedded in the BluReRun 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 (software interrupts) almost simultaneously and serve all of them in real time and in dynamic way. The software architecture is depicted in Figure 4-7.

The BluReRun software architecture is the same for ALL the BluReRun nodes of the Parrotfish project. Upon powering up the Host enters in the idle process. That process makes the μC to enter a power saving mode until a software interrupt is triggered. Any software interrupt can be triggered either by incoming information from USART 1 (meaning the HPT module, layer 2 of the Parrotfish project) or from USART 0 (the Bluetooth module). When that happens, the entire necessary software modules are executed depending on the case.

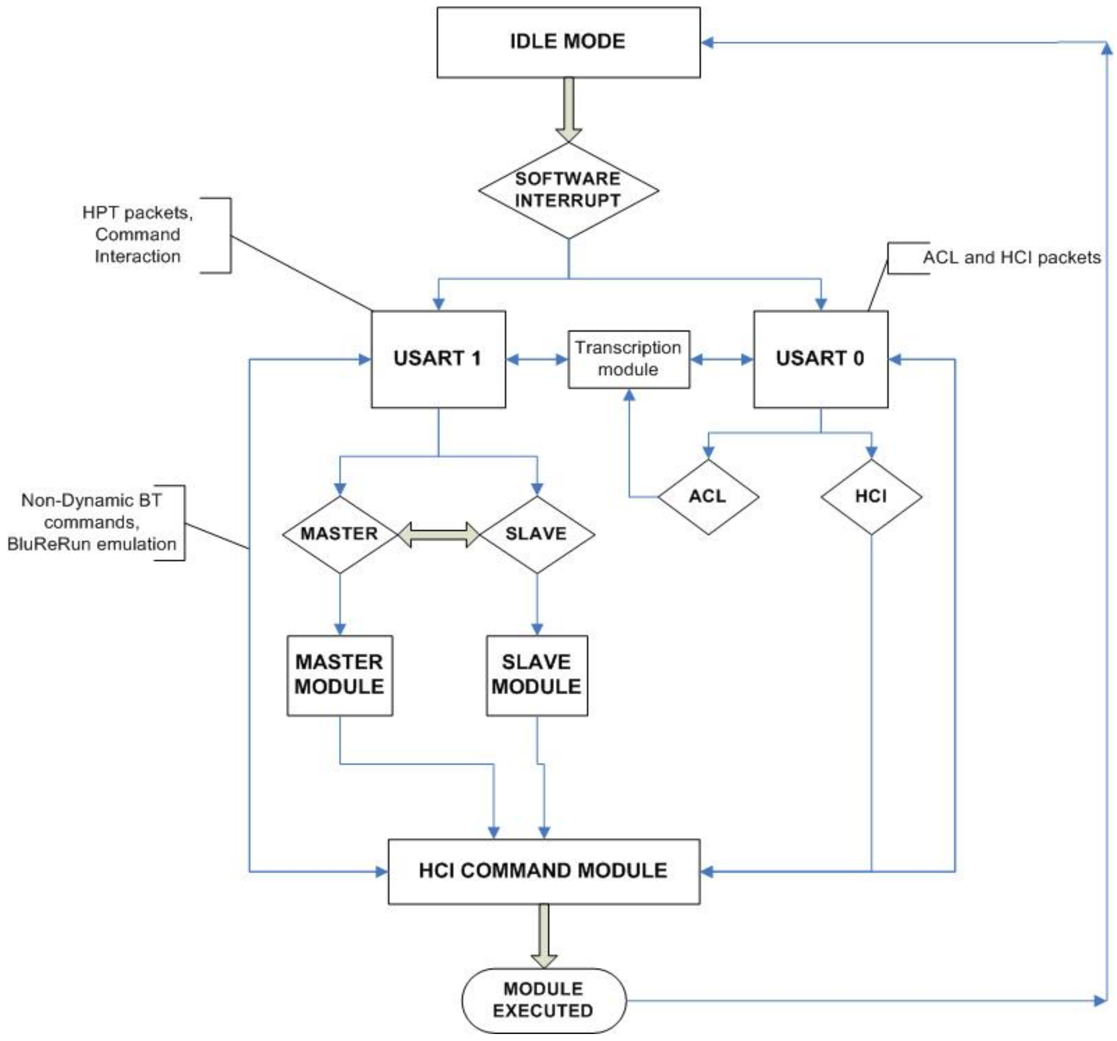


Figure 4-7. BluReRun software Architecture

If the software interrupt was triggered from the USART 1 then the HOST must check whether it must follow the Master or the Slave module routine. This information is given by the HPT through USART 1. Also the HOST might be necessary to alter its status and migrate from master to slave or vice versa. In any case the necessary commands are implemented through the HCI Command Module. Finally in the case of an HPT packet it is necessary either in master or slave mode for the transcription module to be called. If the software interrupt was triggered by the USART 0 then the HOST is checking for either the HCI event or an ACL data packet. 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 BluReRun system to function are the USART Transmit and Receive control modules (USART0 and USART1). They will be specified in the following section.

4.3.1 USART Transmit and Receive control modules

The USART control modules undertake the most important task in the BlueReRun design; they must receive, decode and transmit packets to the BT module or the External device which in the case is the HPT. Errorless and efficient function of these modules is necessary because the design's credibility relies almost solely to these. The system has 2 USARTs, one dedicated to the communication between the μC and the BT module and the other between the μC and an external device which is the HPT. Every USART can receive and transmit bytes. Four software modules, one module for each of the two receivers (USRX) and two transmitters (USTX) have been implemented. (Figure 4-8)

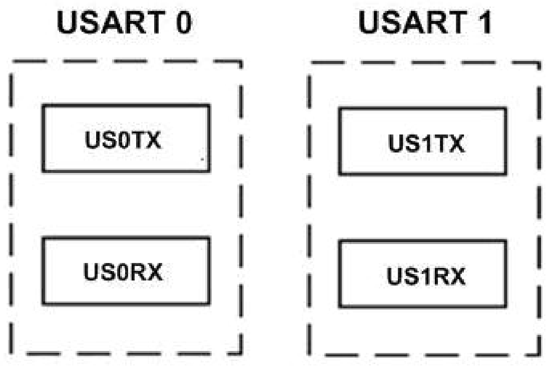


Figure 4-8. USART software modules

In Figure 4-9 the basic idea on the USART Receive and Transmit modules of a certain node is depicted. The HOST (as it will be examined in the next sections) communicates with the HPT and the BT module through USART 1 and USART 0 respectively.

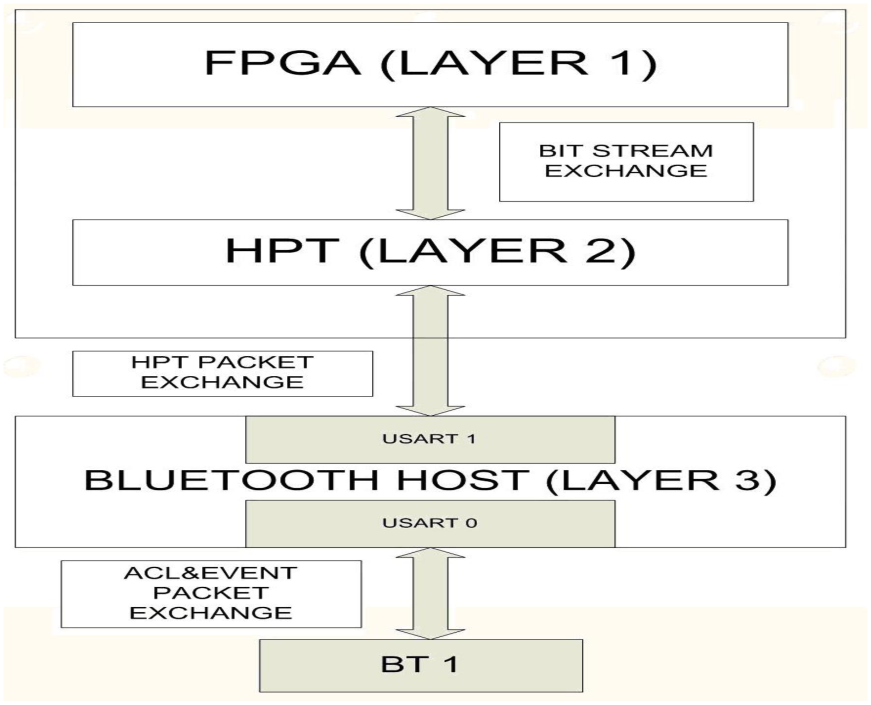


Figure 4-9. The USART architecture

The two USART 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 USART reception decoders

The USART reception decoders (US0RX, US1RX) are the highest level software modules included in the BluReRun system. They run all the Bluetooth protocol layers that are embedded in the system, give Bluetooth interoperability and functionality to the BlueReRun system and enable the essential information exchange (through the HPT packets) between the lowest level of the Parrotfish Project and the upper layers (HPT and FPGA) in an ad hoc network.

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

- The US0RX: It decodes packets being received from the BT module
- The US1RX: 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 flow diagram of the decoders is depicted in Figure 4-10.

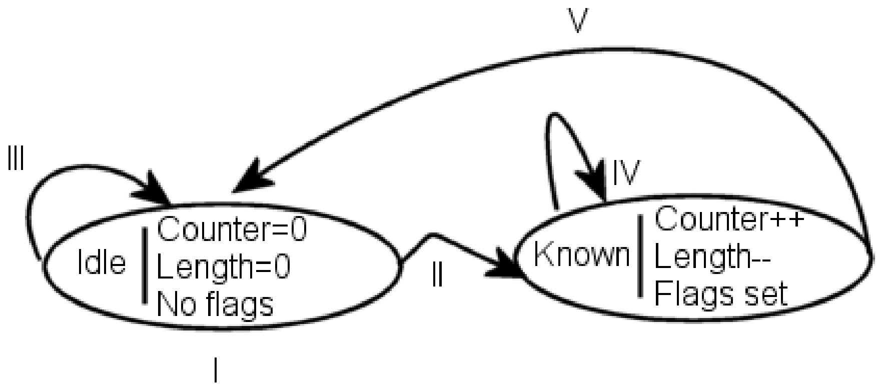


Figure 4-10. USART decoder state diagram

The basic steps of the decoder algorithm follow:

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 2nd 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 BluReRun 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.

The basic decoding algorithm that was used is efficient and errorless since every packet received from either the USART0 or USART1 is correctly transcribed and decoded. Of course, the structure of the decoders is far more complicated than that depicted in Figure 4-10. In the following sub-sections we will see the tasks that each of the USART decoders undertake.

4.3.1.1.1 USART receiver from the BT module [US0RX]

One of the most important tasks that BluReRun 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 US0RX is:

1. To decode the incoming events that the BT module transmits to the Host

2. To transcribe the ACL data packets, which are targeted for the EXT DEV (the HPT or another external device), to the HPT packet format and retransmit them (through USART1) to the HPT.

Decoding of incoming events

By decoding the events, US0RX collects and stores to the HOST SRAM the information about the devices in the Bluetooth module's vicinity and the connected to it devices. It also notifies the EXT DEV (meaning the HPT) for the disconnections or character (master to slave or vice versa) migrations or HPT command packet events, as they happen. Furthermore, US0RX uses the display system to output the success or not of a command issued to the BT module (debugging purposes).

All the events supported in BluReRun are implemented according to the BT spec 2 and as a consequence BT spec 1.2 [3]. For each event decoded, the US0RX takes different actions. Usually, when an event is decoded, the US0RX 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 HOST is used. The upper 4 LEDs indicate the event's identity and the lower 4 show the event's status or other significant event parameters. Not all the events are transmitted through the USART1 to the HPT (or any other external device). The ones that are transmitted to the HPT are transcribed to HPT packets.

The US0RX decoder supports the following events:

Event	**Event code (hex)**	**Length**	**Parameters**
Inquiry complete	0x01	4 bytes	Status
Inquiry result	0x02	18 bytes	# of Responses, BD ADDR, Page Scan Repetition Mode, Page Scan Period Mode, Reserved, Class of Device, Clock Offset
Remote name request complete	0x07	258 bytes	Status, BD ADDR, Remote Name

Connection complete	0x03	14 bytes	Status, Connection Handle, BD ADDR, Link Type, Encryption Mode
Disconnection complete	0x05	7 bytes	Status, Connection Handle, Reason
Command complete	0x0e	(6+n) bytes	# of HCI Command Packets, Command Opcode, n bytes of Return Parameters
Command status	0x0f	7 bytes	Status, # of HCI Command Packets, Command Opcode
# of Completed Packets	0x13	8 bytes	# of Handles, Connection Handle, HC # Of Completed Packets

The following output actions are taken for every event supported:

Event	Output (Upper 4)	Output (Lower 4)
Inquiry complete	2nd LED	# of devices
Inquiry result	2nd LED	# of devices
Connection complete	3rd LED	2nd LED
Disconnection complete	3rd LED	1st & 2nd LED
Remote name request complete	3rd LED	1st LED
Command complete	1st LED	Status byte
Command status	3rd LED	Status byte
# of Completed Packets	4th LED	number

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 or the HPT.

ACL Data packets

The US0RX decoder is the software module that undertakes the task to communicate with the necessary transcription module in order to transcribe the

BlueTooth data (ACL) packets received from the BT module to HPT packet format. After that it retransmits them through the USART1 to the HPT or any other external device.

In all the BluReRun hosts, the ACL data packets received from USART 0 are transcribed to HPT packets (Figure 4-7). Through this method, the transparency and efficiency of the system is secured. The transparency to the upper layers of the Parrotfish system is secured since the only mode of communication between the HPT (layer 2) and the BluReRun platform (layer 1) are the HPT packets. Therefore, each HPT understands that it communicates solemnly with another HPT. The system with this method is more efficient since the only transcription made is from ACL data packets → HPT packets; thereby no large HOST resources are being used. The identity of each received ACL data packet is again checked for validity; if it is not the expected, the ACL packet is ignored or can be stored in an external memory if such is used(in this thesis this was not necessary since the bytes send from one HPT to another were no more that 256) .

4.3.1.1.2 USART receiver from the EXT DEV [US1RX]

Another very important software module of BluReRun is the decoder of the packets coming from the HPT or any other (for example PC) EXT DEV (US1RX). The importance of the US1RX module lies to the fact that it is responsible for securing efficient data reception and transmission between the BluReRun node and the HPT, thus guaranteeing correct FPGA re/configuration.

The USART 1 of the HOST of any BluReRun node can receive the following types of information:

- HPT Data packets to transmit them wirelessly to any BluReRun node through transcription to ACL data packets.

- Non Dynamic Bluetooth commands from the HPT or any other EXT device to the BT module
- HPT-BluReRun Command Interaction. The Flash command interaction describes a control and command protocol between the HPT and the BluReRun.
- BluReRun emulation. Packets indicating a Bluetooth command implemented in the µC to be issued to the BT module. The communication is carried out in 57.6Kbps by default (but also 115.2kbps and 256kbps are also supported), as also happens with the BT module.

HPT Data packets

HPT data packets that are transmitted to the BluReRun by the HPT or any EXT DEV must have the format described in Figure 4-3. After the US1RX detects that a HPT data packet is being received, the following steps will be executed:

a) Transcribe the packet to a Bluetooth protocol compatible ACL data packet of the form of figure 4-4
b) The above procedure will be executed in the Transcription module of the USART 1 through correctly positioning the handle, length and data to the essential ACL format
c) Upon completion of the transcription, the ACL data packet is transmitted through the USART 0 to the Bluetooth module, so that it will wirelessly transmit it to the target BluReRun device (if it exists).

The validity of the handle that has been received, which indicates the target BluReRun node, will be checked by the BT module. Only data packets in the HPT data packet format are accepted by the USART 1. This secures transparency between the different layers of the Parrotfish project.

Non Dynamic Bluetooth commands

One characteristic of BluReRun that makes it more efficient than its predecessors is the dynamic command initialization. This BluReRun trait will be examined in section 4.3.4. At the same time since every BluReRun node can function as an immediate Bluetooth command transmitter, the USART 1 of the HOST accepts non dynamic Bluetooth commands from the EXT device or dynamic Bluetooth commands from the HPT. In the case of any other EXT device these commands have the format of figure 4-11. In the case of the HPT the control and command protocol (HPT-BluReRun command Interaction) is implemented. After the US1RX detects the Non Dynamic Bluetooth Command 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.

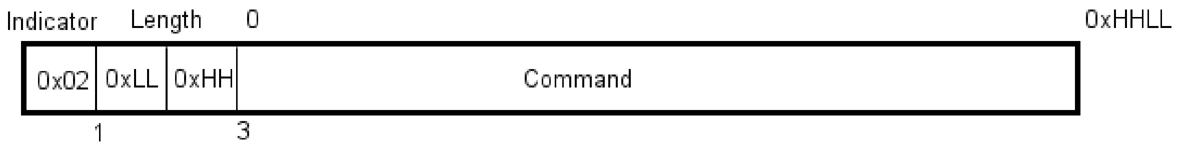


Figure 4-11. Non Dynamic Bluetooth commands from EXT device other than HPT

HPT-BluReRun Command Interaction (Flash Command Interaction)

Another characteristic of the BluReRun communication protocol is the HPT–BluReRun Command Interaction. This command protocol between the HPT (layer 2 of the Parrotfish system) and the BluReRun (layer 1) is based on the idea of dynamic Bluetooth commands. The basic idea of this protocol is that the master HPT, since it is the leader of the whole system, must have a command interaction mode between it and the BluReRun in order to proceed to inquiries, connections, disconnections or master to slave migrations. This protocol states that whenever the master HPT transmits to the BluReRun host an HPT packet in the form of Figure 4-12, then the BlueReRun host US1RX module understands that the data that follows must be read from him as it is a command.

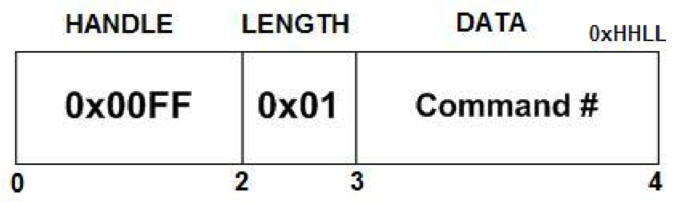


Figure 4-12: HPT command Packet

In Figure 4-12 the HPT command packet is depicted. The HPT command Packet has the following components:

a) The Handle: This is always 0x00FF. It is the code that states to the HOST that what follows is a command for the Bluetooth module. This handle is unique for the HPT and the BluReRun.

b) The Length: It is always 0x01 since the data that follows describes a command that is embedded in the flash of the Host and the HPT.

c) The Data: The data is 0, 1, 2, 3. These numbers correlate to the position of the command in the flash memory. The HPT as well as the HOST have those four commands in the same Flash memory position, as depicted in the following table. This characteristic makes the whole command protocol dynamic. The HPT does not need to send a command in the Non Dynamic Bluetooth commands format.

HCI command	**Details**
0) *Inquiry command* Data: $00	<u>Command format:</u> $01$01$04$05$33$8b$9e$05$00 Searches for active Bluetooth devices in the vicinity and returns relative information
1) *Create Connection command* Data:$01	<u>Command format:</u> $01$05$04$0D$(6 bytes of BD_ADDR) $18$cc$(page scan repetition mode)$(Page scan mode)$(Clock offset) $(Clock offset) $01(role switch enabled)

	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
2) *Disconnection command* Data: $02	<u>Command format:</u> $01$06$04$03$ $$(connection handle of the device we want to disconnect) $13 (reason) Terminates an existing ACL connection based on the connected device's Connection Handle
3) *Master Slave Switch command* Data: $03	<u>Command format:</u> $01$0b$08$ (BD_ADDR)$00 from master to slave $01$0b$08$ (BD_ADDR)$01 from slave to master Used to switch role of the BT module from master to slave or vice versa.

In order to understand the HPT – BluReRun command interaction module it is essential to describe an example. Let's assume that the HPT would like to know the number and the names of other HPTs that are in its vicinity. In order to do so it must issue an inquiry command through the Bluetooth module. Thus, through HPT – BluReRun Command Interaction it transmits to the BluReRun Host the HPT command packet: $00$FF$01$00. The USART 1 decodes the HPT command packet received, and transmits to the Bluetooth module through USART 0 the command in flash position #0. This command is the inquiry. The event returned is transmitted in the format of HPT data packets to the HPT. The same procedure happens with the other two commands. This command communication protocol is very efficient for the following reasons:

a) The HPT does not need to know all the commands of the Bluetooth protocol. Only those three that are necessary for the HPT are implemented.
b) The resources of the HPT are very limited. If all 11 commands of the BluReRun Bluetooth protocol were implemented then the FPGA programming (which is the HPT task) would be in high risk.
c) The transparency of the whole Parrotfish project is secured since the communication between the HPT and the BluReRun is done in HPT packet format.

BluReRun Emulation

The BluReRun emulation is a BluReRun characteristic that does not apply to the HPT layer. It applies, however to any other external device such as a PC. The BluReRun emulation packet is depicted in figure 4-13. It consists of a packet indicator and 3 more bytes.

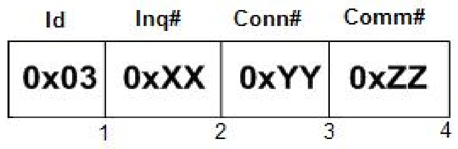


Figure 4-13: BluReRun emulation packet

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 BluReRun to be executed by it.

This module has been designed in order to help any BluReRun node to be able to function without the explicit need for a human user. It is based on the HPT-BluReRun command interaction.

4.3.1.2 USART Transmission Module

The Bluetooth module communicates with the HOST through the USART 0. The HPT or any other external device also uses this way to exchange data with the

HOST and/or control it through USART 1. The predecessors of BluReRun did not have an efficient buffering mechanism (actually only the BluMiu had one, but was rather inefficient). Therefore, the creation and implementation of an efficient buffering mechanism was essential. In Figure 4-14 the buffering mechanism and algorithm is depicted.

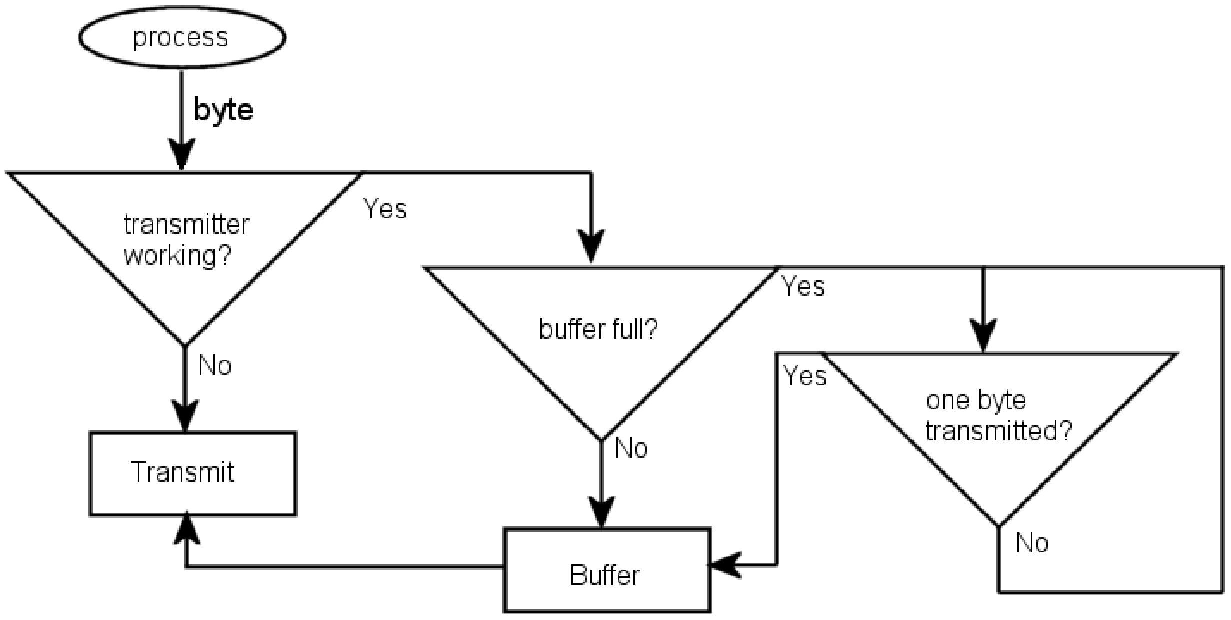


Figure 4-14: BluReRun transmit buffer and mechanism

Two 256byte buffers are allocated in the HOST's SRAM, one for every USART. The size of 256 bytes was chosen since the HPT forwards through the BluReRun data of 256 bytes (in the form of bit stream files) for the configuration of the FPGA. By using the buffering mechanism depicted in Figure 4-13, when an application requests to transmit a byte from a USART, the mechanism first checks to see if this USART is in the process of transmitting another byte.

- If the USART 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 USART 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.

- o If the buffer is not full, the new byte is stored in the buffer so that the USART will transmit it when it has finished transmitting all the bytes that are stored in the buffer before that byte.
- o If the buffer is full, the USART mechanism will be stalled until a byte from the buffer has been transmitted. Until then, the new byte is stored in the HOST's memory along with other data essential to the USART 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 USART finishes transmitting a byte, a USART transmit complete interrupt is triggered (TX). In the TX interrupt service routine (ISR), the USART mechanism checks if the buffer is empty.

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

The USART buffering mechanism was implemented only for the transmission modules. Buffering did not need to be implemented in the receivers because the HOST is fast enough so that it can process a received byte completely before the next byte arrives in the USART receiver, even if the system is manipulating byte streams belonging to two different packets at the same time (for example an HPT packet and an ACL data packet). The maximum CPU time for a decoder to fully decode a byte that has just been received is 23.6usec[17] (the last byte of the header of an incoming BluReRun data packet from the HPT, which needs to be transcribed to a Bluetooth ACL data packet, so that it will be transmitted by the BT module to a connected device). Multiple interrupts can be processed by the HOST, while the decoders process a byte. Though, all the ISRs implemented in

[17] Most of the commands implemented in the AVR need a single clock cycle to execute [11] (even for SRAM transfers) and an 11.0592MHz clock is used, so the AVR's throughput is approximately one command every 9nsec

BluReRun complete in a maximum time of 23.6usec and a nominal time of 4usec. Two consecutive bytes are received from the HOST's USART 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 USART. The HOST, either way, can wait with one byte in its data reception register for some μsec, until the decoders end their jobs.

4.3.2 Master – Slave Integration in the BluReRun HOST

One very important contribution of this thesis is the integration of master and slave Bluetooth characteristics in the BluReRun host. In the Bluetooth network topology there is one master and many slaves (section 2.3.2). What differentiates the master with the slave modules is its ability to send inquiries, create connections and adjust the disconnection process. In every Bluetooth piconet or scatternet there is a master that controls the communication between it and the slaves, as well as creates the addressing protocol which is vital for efficient data exchange [2]. Every BluReRun node must be able to act as a master or a slave module. When we have the trivial case of only two nodes in the ad-hoc network or when we have 256 nodes then one of them will be the master and the others the slave. In the Parrotfish project each node has the format of figure 4-1. The HPT (layer 2) is the leader of the system and thus the one that states whether the node is a master or a slave. If the Parrotfish node is a master then the BluReRun Host must execute the master software module. If the Parrotfish node is a slave then the BluReRun Host must adapt accordingly. In any case the BluReRun host waits for this software interrupt that will trigger the execution of either the master or slave module.

In the following table and in Figure 4-15 the algorithm steps and the flowchart are depicted.

```
Step 1: Idle mode
Step 2:     Software interrupt
      HPT master-slave switch command packet is received from USART 1

Step 3: If (master == true) then
            {Execute the master module
            Inform the HPT through HPT command packets when necessary
                    When terminated ➔ return to idle mode}
      If (slave == true) then
            {Execute the slave module
            Inform the HPT through HPT command packets when necessary
                    When terminated ➔ return to idle mode}
      Else
            Ignore and return the HPT packet back to the HPT
            Return to idle mode
```

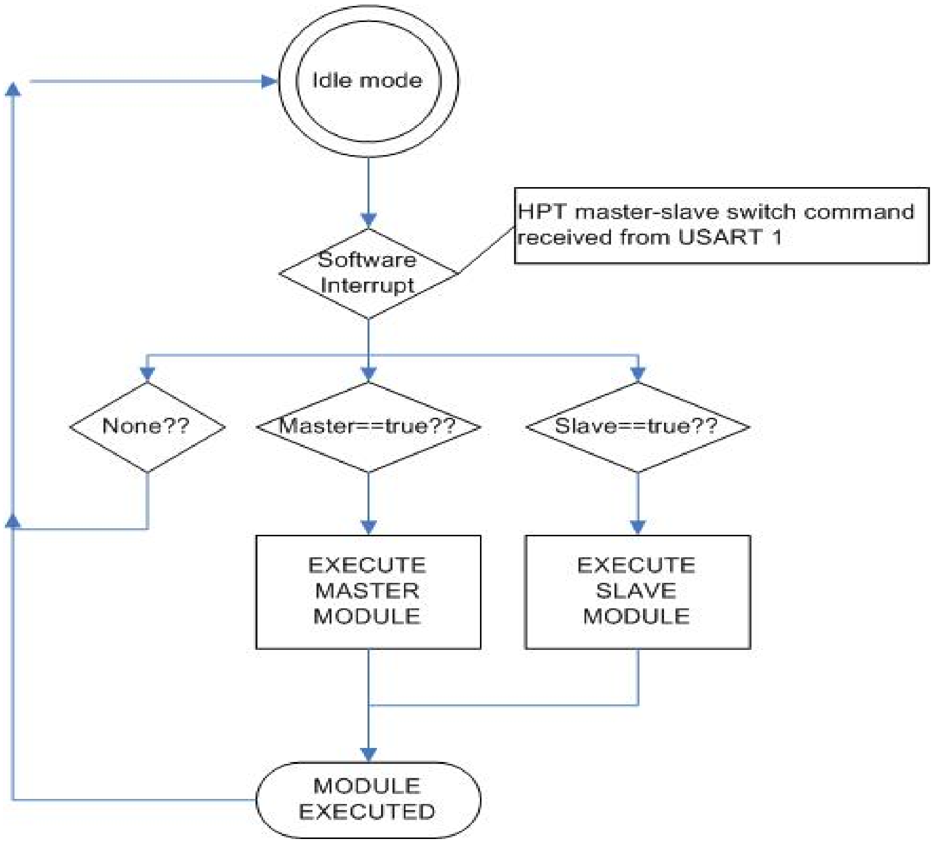


Figure 4-15: The Master to Slave migration and vice versa

The BluReRun HOST has all the Master and Slave modules embedded in its software architecture. When the Master Slave switch command is received from USART 1 then some basic functions are taking place. This simple algorithm guarantees the efficient master to slave or vice versa migration in a piconet or scatternet.

4.3.3 Dynamic Command Initialization

Another important BluReRun contribution is the dynamic command initialization. The purpose of this characteristic is to improve the performance of each BluReRun node over the existent architectures and to create an efficient dynamic BluReRun node. The basic idea is to create an algorithm where every node either a master or a slave would be able to execute certain steps that would lead to dynamic command initialization and configuration. These steps correlate to specific commands that are essential for the existence of the Bluetooth Protocol. Thereby, 4 commands that are of importance to the function of the Bluetooth protocol in either point to point or point to multipoint architectures were implemented in the Dynamic Command initialization method.

a)	**Inquiry command for the master node**
b)	**Create Connection command for the master node**
c)	**Set event filter command for the slave node**
d)	**Write Scan Enable for the slave node**

With the use of Dynamic Command Initialization (DCI) each BluReRun node does not need to be issued any command necessary for discovery, or connection. Everything is done in a dynamic way. The commands described in the above table were not selected randomly. The first two commands are essential for the Bluetooth master node, and they are those commands that differentiate the master from the slave. They are important for discovering other nodes in the vicinity and connect to them. The last two commands are necessary for each Bluetooth slave. They activate the Bluetooth module mechanisms that are responsible for making the slave BluReRun node visible from the master one. In the following table the basic steps of the algorithm follow:

```
STEP 1: BluReRun host powers up and the main routine initiates in both the master and the slave
STEP 2: In the Main routine the issue of the command Reset is executed. The issue of this command is
        necessary for the powering up of the Bluetooth module
STEP 3: If (master ==true) then
        {If (reset event == true) then
              {execute the Inquiry command
              When Inquiry result event is returned from the Bluetooth module
              Then call from the Flash memory the Create Connection command.}
        }
        Elsif (slave==true) then
        {If (reset event ==true) then
              {execute the set event filter command
              When command status event is returned from the Bluetooth module
              Then call from the Flash memory the write scan enable command}
        }
STEP 4: Terminate the algorithm and return to idle mode.
```

The BluReRun host is using the indexes of the commands that are embedded in the Flash memory of the microcontroller. The idea is the same as the BluReRun emulation and the HPT-BluReRun command Interaction.

4.3.4 Addressing Protocol Algorithms

The BluReRun system, as it is so far understood, can work either on its own or as a sub-system in the Parrotfish project. Of course, its ultimate target is to provide an errorless and transparent communication medium for wireless FPGA Re/configuration. Every Bluetooth system must support either point to point or point to multipoint addressing protocols. The importance of the addressing protocols in piconets or a scatternet is large, since this protocol will guarantee an efficient packet transport from the destination node to the target one. At the next two sections the addressing protocols for point to point as well as point to multipoint architectures will be examined.

4.3.4.1 Point to point addressing protocol

In the case of point to point communication the addressing protocol is of trivial importance. Since in the ad hoc network there are two nodes, one master and one slave, no significant addressing protocol is required. In the case of the Parrotfish system the HPT packet communication protocol is sufficient. Let's assume that we have a system of two Parrotfish nodes as depicted in Figure 4-16:

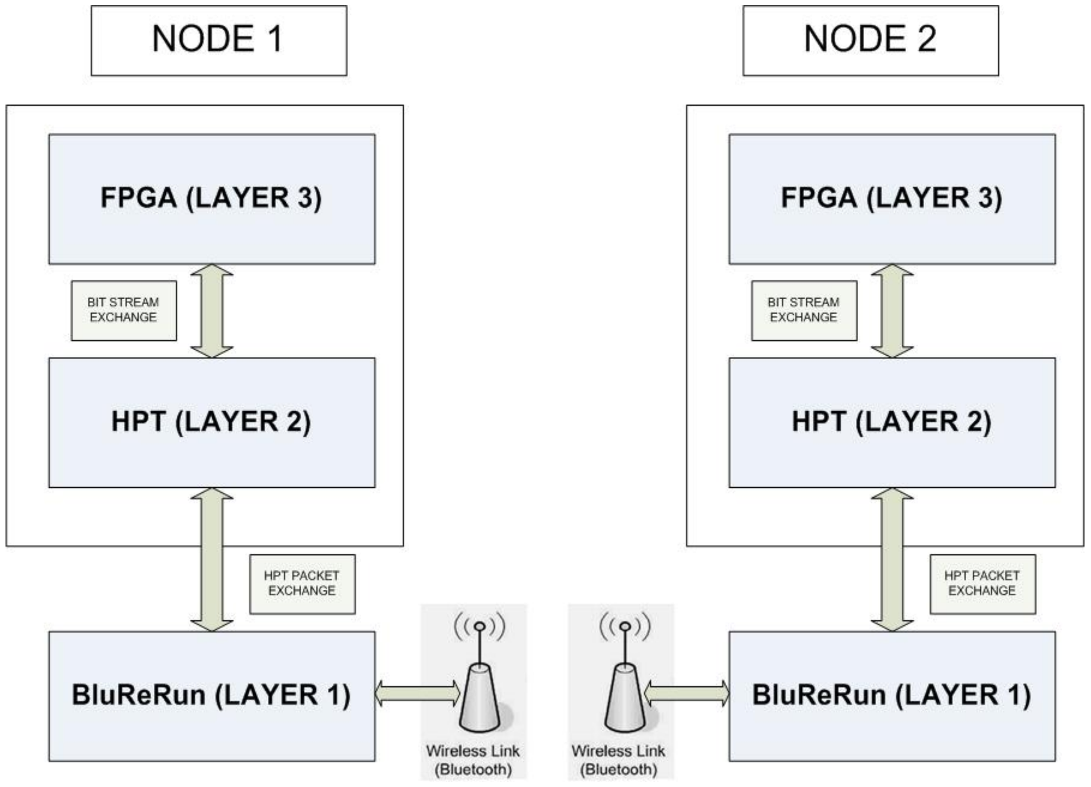


Figure 4-16: The Parrotfish point to point architecture

In the case of Figure 4-16 the master node is the NODE 1 and the slave node the NODE 2. The HPT packet format and the ACL data packet format secure the efficient data or control packet exchange between those two nodes. Also, since we have only two nodes the handles will be fixed and do not need to alter at any circumstances. Even if the HPT who is the leader of the system, decides to migrate from master to slave or vice versa the master-slave migration algorithm will be implemented and all the modules will be executed accordingly.

4.3.4.2 Point to multipoint addressing protocol

The creation of a point to multipoint addressing protocol that will coincide with the Bluetooth specification version 2.0 and the Parrotfish's project requirements was the most difficult task in the BluReRun thesis. The point to multipoint addressing protocol must be able to:

a) support an efficient and errorless data and command packet exchange between the different nodes of the ad hoc network
b) adjust to any potential difficulties or changes that the ad hoc network might face (for example low data rate due to high trafficking, high error rates, master to slave migration) and,
c) retain the transparency mode between all the levels of the Parrotfish

In order to fulfill the above three requirements an efficient algorithm was created. This algorithm is based on the fact that the HPT is the leader of the Parrotfish project and on the fact that the transparency between all the levels of the node is secured. Before examining the algorithm it is vital to understand the use of the Read HPT name command.

Read HPT Name Command

This command is a unique command between the HPT and the BluReRun levels of the Parrotfish project. This command was introduced from the need for the BluReRun level to correlate correctly the handle of the HPT packet with the HPT name. The handle of the HPT packet is not given by the HPT. It is given from the Bluetooth module firmware. When a create connection command is issued, a create connection complete event is returned [4]. In there lies the handle of the BluReRun nodes that lie in the vicinity. At the same time, each HPT has a unique name (HPT01, HPT02 etc). Therefore, in order for the BluReRun host to correlate correctly the HPT name to the handle of the HPT packet the Read HPT command was created. The format of the command is based on the HPT-BluReRun command Interaction and it is depicted in Figure 4-17.

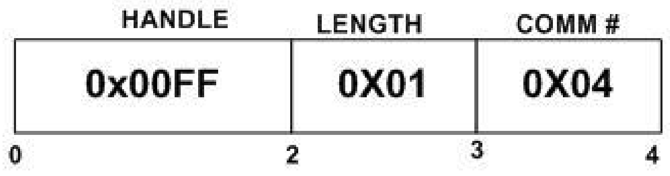


Figure 4-17: Read HPT Name Command

The COMM# field is 0x04. This number correlates to the position of the Read HPT Name Command in the Flash memory of the BluReRun HOST. When the Read HPT Name command is issued from the BluReRun HOST to the HPT then the HPT transmits to the BluReRun host's USART 1 its name. The HPT's name may be HPT01, HPT02 etc. After that the following point to multipoint algorithm is executed:

Point to multipoint algorithm.

Step A:

The BluReRun HOST sends to the HPT the Read HPT name command, which the HPT understands that it is a command for him. In this command the HPT immediately sends its name in the form of 'H''P''T''-''0''0'1'. The 1 maybe will change to 2 etc. (meaning'H''P''T''-''0''0'2'). When the BluReRun host receives that packet then it decodes it, and saves to a memory address that is created for that purpose the name (HPT-001, HPT-002 etc).

Step B:

Inquiry command issued by Master BlueReRun node after such a command is requested by the HPT (through HPT-BluReRun command interaction). For example the HPT master would like to know the status of other HPTs in the Parrotfish ad hoc network, their names in order to move to connection, disconnection and other data transactions.

Step C:

Inquiry result received by the master node. In there lies one or more of the BD_ADDR of the BT modules. The BlueReRun host stores all the BD_ADDR. When an inquiry result event is received then the BlueReRun host sends a create connection command. Then he receives the connection complete event where the handles lie. The BlueReRun host stores all the handles in the same memory directory as the BD_ADDR that correspond to those handles.

Step D:

For every BD_ADDR the BluReRun host does Remote Name Request so that it will know the exact correspondence between BD_ADDR and the name of the HPT where from step A the other

BluReRun host already knows. Therefore, in every transaction the BlueReRun host will know exactly the BD_ADDR, handle of the BT module and the name of the HPT.

Step E:

The BluReRun host transmits to the HPT the whole list of Handles and HPT names in the order of: \$00\$01 → HPT01, \$00\$02 → HPT02 etc. This is done through HPT packets and it does not affect transparency since the HPT packet communication format is kept.

Step F:

The master HPT will then know which other HPTs are in its vicinity and send the appropriate HPT packet for the BluReRun to forward to every slave it needs.

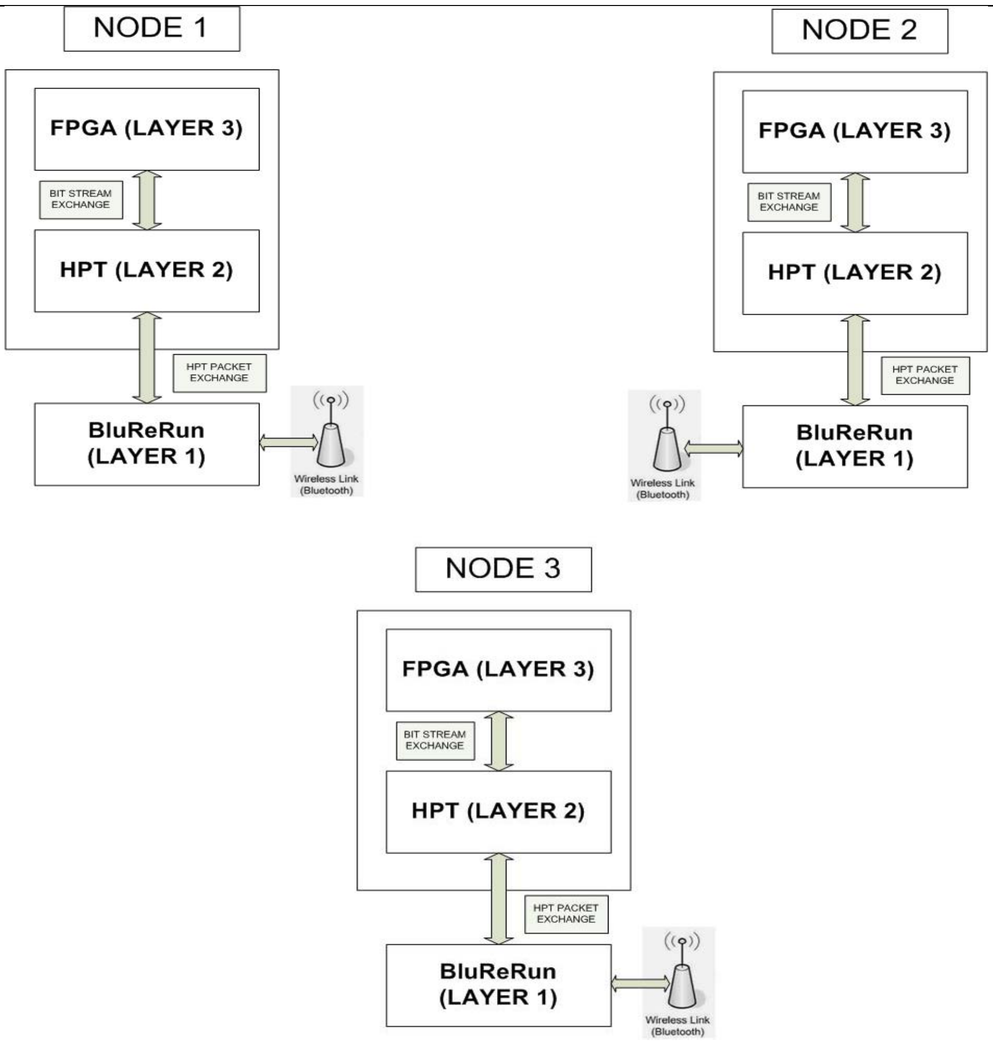


Figure 4-17: The point to multipoint implementation

In order to understand the above algorithm it is imperative to examine thoroughly its implementation in an example. Let's take the Parrotfish nodes of Figure 4-17 for example. In that figure there are three nodes that have the format of the Parrotfish project. Let's assume that the node 1 is the master and the others the slaves. This case study will be examined from the BluReRun point since implementation of layers 2 and 3 of the Parrotfish node are not in this thesis' scopes. At first, upon initialization, the BluReRun host transmits a Read HPT Name Command to the layer 2 (HPT). Then the HPT transmits its name in the format of STEP A of the algorithm. Afterwards, the layer 2 (HPT) of master node (Node 1) would like to know the other HPTs that in its vicinity and their status. Thus through HPT-BluReRun command Interaction it issues an Inquiry command to the BluReRun (STEP B). The BluReRun HOST will execute the necessary command and a response will be returned. When the response is received then the BluReRun HOST will issue a create connection command. The BD_ADDR and the handles of the Bluetooth modules will be stored in the HOST SRAM (STEP C). The next step is for the master BluReRun host to issue the remote name request command. The remote name request event is received and the BluReRun host saves to its SRAM the Handle, BD_ADDR and the HPT names (STEP D). In STEP E the BluReRun HOST transmits in the form of the HPT packets the handles and the correlating HPT names. Therefore, in STEP F the master HPT (and therefore the master node) knows the HPT names and the handles of the HPTs that are in its vicinity. After that, all the functions that the BluReRun system accepts can be executed.

After the through examination of the BluReRun addressing protocol, it is important to study the error handling routines that were implemented in the BluReRun thesis.

4.3.5 Error handling Algorithm

When a system is being developed it is vital to create some methods that will guarantee the effective system response to any error or bug related phenomena.

```
STEP 1: The first byte of the handle is received
If (first header byte == valid)
    Then execute the US1RX or US0RX routine and go to STEP 2
Else
      {Discard or send back the byte to the HPT or the BT module. And do not execute any routines or
      set any flags or increase any counter; just wait for a valid header to come( timer counter).
      If (valid header does not come)
          {Then exit to idle mode}
      }
STEP 2: The second byte of the handle is received
If (second header byte==valid)
    Then continue execution of US1RX routine and go to STEP 3
Else
      {Reset all flags and counters. Discard the whole packet. Wait for the first byte of the handle to
      come again (timer counter). Go to Step 2}
STEP 3: The Length byte is received
When the Length byte is received, the initial length is stored in a register.  For every byte that comes the
length is decremented by 1.
If (length ==0)
      Then the HPT packet is finished. The data will be transcribed and forwarded. Return to idle mode
Else
    Then timer/ counter initiated and wait. If the length is not received then exit, discard the packet, reset
    all the flags and return to idle mode
```

Thus, in the BluReRun system an error handling algorithm was created to guarantee efficient data transport that would aim to errorless FPGA re/configuration. This algorithm is executed in the USRX routines of the BluReRun HOST. The idea is that the possibility of an error is far higher in the HPT to BluReRun communication than the Node to Node communication with the Bluetooth protocol. The Bluetooth protocol has many CRC algorithms that are executed whenever the ACL data packets are transmitted from one BluReRun node to the other. The transmission routines are not vulnerable to errors since the transmit buffer is working in a specific mode. Therefore, it is essential to create an error handling algorithm in the US1RX and US0RX module of the BluReRun Host. It is vital to note that the US1RX module as explained in section 4.3.1, accepts HPT packets. The HPT packets are in the format of: Handle (2 bytes) – Length (1 byte) – Data (256 bytes). The ACL data packets are depicted in Figure 4-4. The error handling algorithm is concentrated in the Handle and the Length bytes. The highest possibility of error is done during the transcription from HPT data packets to ACL data packets and vice versa. Thus, the same algorithm is used in the US0RX routine. The Flowchart of the Error Handling Algorithm follows:

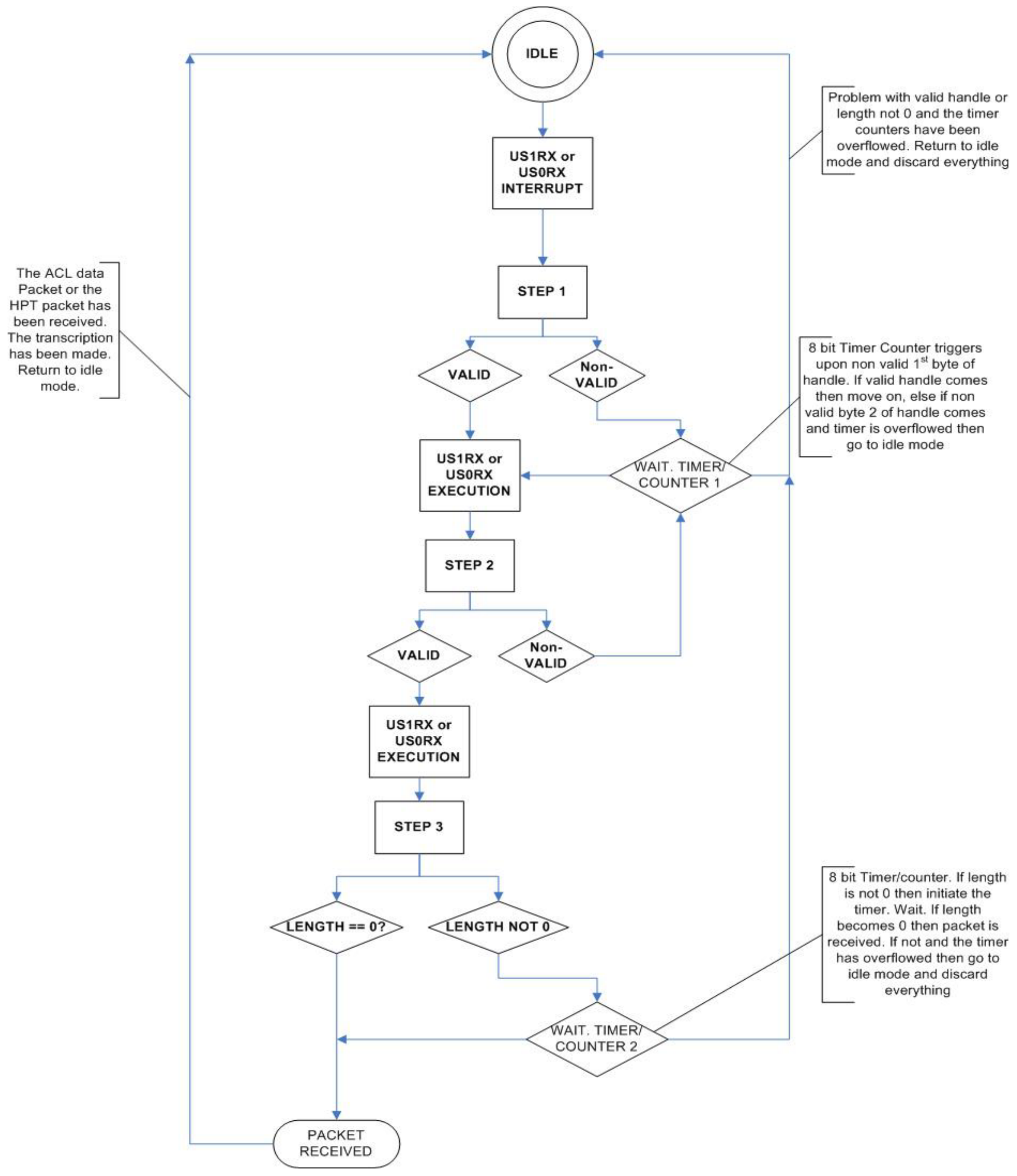

IDLE
US1RX or US0RX INTERRUPT
STEP 1
VALID
Non-VALID
US1RX or US0RX EXECUTION
WAIT. TIMER/ COUNTER 1
STEP 2
VALID
Non-VALID
US1RX or US0RX EXECUTION
STEP 3
LENGTH == 0?
LENGTH NOT 0
WAIT. TIMER/ COUNTER 2
PACKET RECEIVED
Problem with valid handle or length not 0 and the timer counters have been overflowed. Return to idle mode and discard everything
The ACL data Packet or the HPT packet has been received. The transcription has been made. Return to idle mode.
8 bit Timer Counter triggers upon non valid 1st byte of handle. If valid handle comes then move on, else if non valid byte 2 of handle comes and timer is overflowed then go to idle mode
8 bit Timer/counter. If length is not 0 then initiate the timer. Wait. If length becomes 0 then packet is received. If not and the timer has overflowed then go to idle mode and discard everything

Figure 4-18: The Error Handling Algorithm flowchart

The Error Handling algorithm that is depicted in Figure 4-18 is embedded in both the US0RX and US1RX routines. At the same time, any hardware errors that might be created can be detected with the use of the LEDs system. Finally, in the case where a BluReRun HOST is not powered up correctly or if for any reason is damaged then the BT module will not be functioning correctly or not at all (since the BT module powers up from the BluReRun HOST). If this occurs then the BT protocol will initiate a series of “are you alive” commands. If the Bluetooth module does not respond and a timeout happens, then this BluReRun node is considered to be inactive and will be omitted from the ad hoc network.

Having examined the hardware and software parts of BluReRun in detail, the testing procedures used to validate the BluReRun hardware and software design will follow.

5. BluReRun 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 it works properly. Also, during the construction of a complex system, such as BluReRun, 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 function on the boundaries of the Parrotfish project must be examined and its purpose must be secured
- Verification of the whole BluReRun design on the grounds of wireless FPGA re/configuration in an ad hoc network must be, if possible, completed.

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 BluReRun, 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 BluReRun system the ability to use each subsystem as a component, without any doubts if its function was correct or not. Finally, since the BluReRun system is a system that can work on its own and at the same time function as a subsystem in the Parrotfish project, this chapter studies both the experiments that were executed on the BluReRun (alone with the use of a PC) and the experiment that were a combination of BluReRun and all the upper Parrotfish layers (BluReRun and HPT and FPGA) that led to the wireless FPGA re/configuration.

5.1 BT module validation

Most of the components forming the hardware around the HOST (the BT module, the LEDs and the HCI interface) were inherited from the BluMiu system and, thus, did not need extensive tests, except a simple verification that they do function as expected. However, 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 BluReRun by them would be fulfilled, they were exhaustively tested before starting the implementation of the system. The tests for the BT modules were necessary because it was the only component that could provide Bluetooth connectivity and doubts whether its operation was infallible had emerged during the BluMiu development. Another issue that was also tested was if it could offer the services introduced by BluReRun, during the development of the latter.

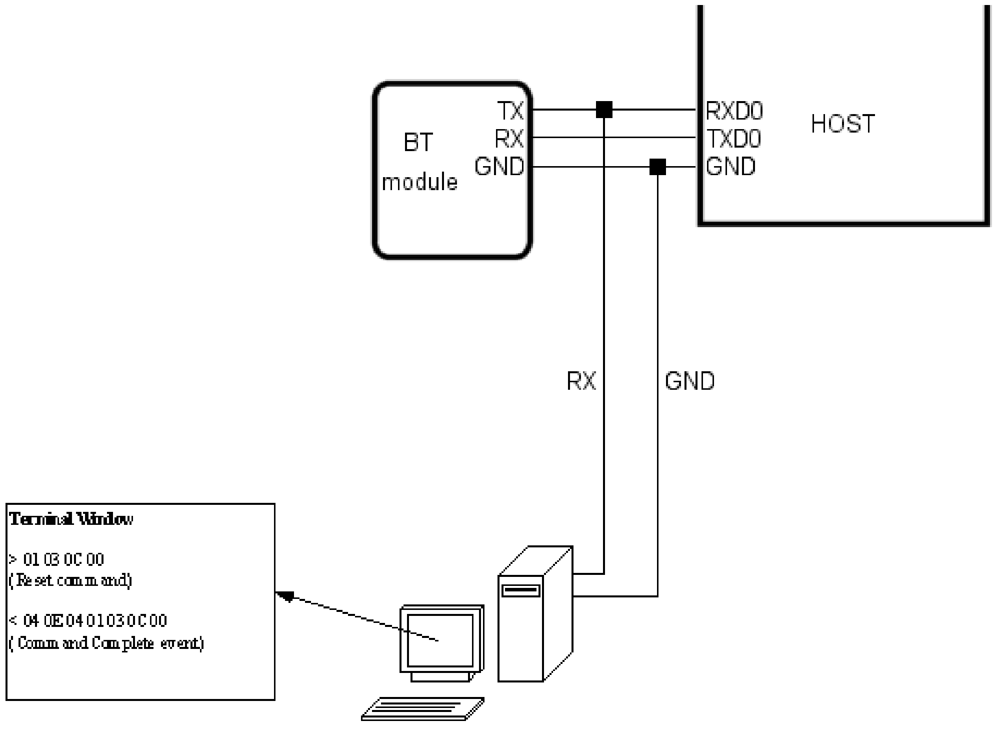
The BT module, as well as its USART interface to the HOST (HCI), was put under trial in the context of the following test procedures:

- By connecting a computer and the BT module through the USART (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 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 PDA (HP7010) as well as a Laptop (DELL Latitude D810) that supports the Bluetooth protocol. 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 460.8kbps data rate that the BT module specifications claim to support was validated using its USB interface and through the connection of the PC and an ADM202EAN with the Bluetooth module. In the case of the 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 TELECA COMTEC.

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



```
> 01 03 0C 00
(Reset command)

< 04 0E 04 01 03 0C 00
(Command Complete event)
```

Figure 5-1: The Bluetooth module and HOST communication

Then, the Bluetooth module and HOST communication had to be monitored. 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 2 (Figure 5-1). In the above figure the RESET command ($01$03$0C$00) is sent from the PC terminal to the HOST and from there to the Bluetooth module. The HCI event that we are expecting to see is the same as specified in Bluetooth specification protocol v2.0 [3].

The software architecture validation is examined thoroughly in section 5.2.

5.2 BluReRun Software Validations

The software design of the BluReRun system is very complex and 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 USART Transmitter Validation

The ultimate purpose of the BluReRun thesis is to create an efficient data communication medium that can work alone in an ad hoc network (BluReRun node) or as a part of the Parrotfish system for wireless FPGA re/configuration. The USART Transmitter is one of the most important modules. The buffering mechanism that was implemented underwent many changes and needed constant debugging throughout this thesis' implementation.

In order to debug the USART mechanism a USART 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 USART 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.

USART 1 (that transmits data to the EXT DEV and the HPT) has an identical mechanism with USART0, so monitoring on it was not explicitly needed. After the USART transmitters' operation was verified, the next most frequently module used was the USART Decoder module.

5.2.2 USART Decoder Validation

The incoming packets from the BT module and the EXT DEV (HPT or PC) are decoded by US0RX and US1RX decoders respectively. The tests to validate the correct function of these decoders used the LEDs and the USART connection to the PC or the HPT 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 BluReRun (according to BT spec 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 US0RX 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 BluReRun port the Connection Handle and the BD ADDR of the device connected to the BluReRun.

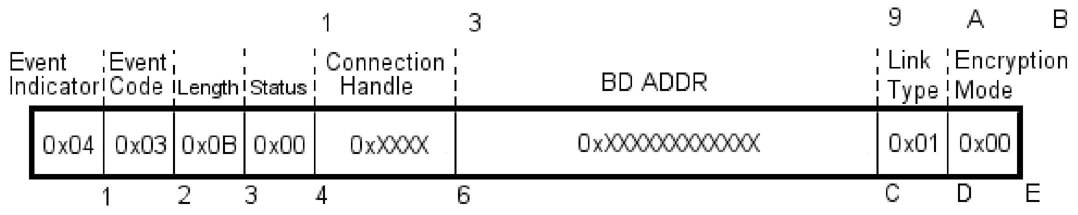


Figure 5-2: Connection Complete Event

The same test procedure was used for the US1RX and the US0RX decoders implemented in BluReRun. Of course, the ultimate test for the transmitters and the decoders would be the successful data exchange between two or more connected BluReRun nodes. This would also be the validation of the whole BluReRun protocol and system that was designed. The test methods that were used to validate the BluReRun protocol were based on the test procedures mentioned in the previous sections and will be presented in the next section.

5.2.3 BluReRun Validation

Through validating the subsystems composing BluReRun, the system itself was tested. The validation procedure of the integral BluReRun 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 BluReRun. This final test procedure was very important for the BluReRun software and hardware architecture. The

BluReRun validation was initially executed in two nodes (master and slave) and then in a point to multipoint environment (three nodes). More specifically the following capabilities of the system were validated:

1. The pre-connection setup of the BluReRun master and slave node using dynamic command Initialization
2. The discovery of various Bluetooth devices by the master and the identification of the slave(s) to the EXT device (PC) or the HPT (upper layer of the Parrotfish project)
3. The establishment of a Bluetooth connection between the master and the slave in either point to multipoint or point to point environments
4. The efficient exchange of data between BluReRun nodes using the BluReRun protocol. Here in the beginning a point to point architecture was tested and afterwards, a point to multipoint. Also an evaluation of its efficiency and comments on possible improvements on the protocol itself were made.
5. The migration from master to slave when requested by the HPT was also validated

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

5.2.3.1 BluReRun Pre-Connection setup

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

The power-up of the BluReRun slave and master node 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 EXT DEV (HPT-BluReRun command Interaction, HPT Data packets, Non Dynamic Bluetooth commands and BluReRun emulation). The BT module is powered up along with the HOST, since it is powered by the BluReRun platform, and according to its datasheet there is no need for a power up sequence [14]. After the power up of the BluReRun 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: BluReRun-001 to the slave and BluReRun-002 to the master)
- The slave should be visible in an inquiry scan. This is done by executing the Write Scan Enable command implemented in the BluReRun HOST using dynamic command initialization
- The slave should set the auto-accept connection parameter of the BT module for every device, so that it would be able to connect with any master or another Bluetooth device automatically. This is accomplished by executing the Set Event Filter command implemented in the BluReRun HOST using dynamic command initialization.

The commands used above should be able to be replaced (another name can be given, the master too can be scanned in an inquiry and different event filters can be set for both the master and the slave[18]) and/or other (not implemented in the HOST) HCI commands can be issued to the BT module by using the Non Dynamic Bluetooth 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 BluReRun slave 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

[18] For information on the functions of the Write Scan Enable and Set Event Filter HCI commands, refer to the BT spec 2. [3]

accomplished using some Bluetooth enabled PDAs and laptops (DELL Latitude D810, HP4010).

Validation of the Non Dynamic Bluetooth commands and BluReRun emulation modes was achieved by enabling the master to be scanned in an inquiry, by executing the commands through the BluReRun emulation and by executing the Read Supported Features and Read Buffer Size commands to the slave. These commands' successful execution was validated through the USART monitoring of the Command Complete events that were returned for each one of them.

Finally, validation of the HPT data packets and the HPT BluReRun command Interaction was executed by the lighting of the correct LEDs when the HPT data packet was received by the BluReRun Host and also through the terminal of the PC that supported Bluetooth interface.

5.2.3.2 BluReRun Discovery of other Bluetooth nodes

After correctly starting-up the BluReRun system either in dynamic or non dynamic mode, the master should be able to search its vicinity for Bluetooth devices, identify and connect to them.

By issuing the Inquiry command to the BT module from the BluReRun host, the BT module will report the Bluetooth devices it found to the HOST with an Inquiry Result event for every device[19] it found. The master 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 as well as handle

[19] BT spec 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.

through implementation of the point to multipoint algorithm to the EXT DEV or the HPT through the USART1. The user friendly name of an inquired device would be 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 master HOST should store this name in its SRAM and forward necessary information regarding other BluReRun nodes in the ad hoc network either in point to point or point to multipoint environments.

The BluReRun master's ability to locate Bluetooth devices and identify them was validated by finding any slave and reporting its characteristics through the EXT DEV. The master's ability to support many inquired devices and report the selected device characteristics was validated by locating the other two BluReRun slave nodes and retrieving their BD ADDR, as well as reporting their user friendly names when they are requested for each device.

5.2.3.3 BluReRun master to slave connection

After the master 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 slave, though the slave can't be able to request a connection from a Bluetooth device, only accept connection requests. The master BluReRun node functions also a lower level of the Parrotfish project (data communication medium). The connection between a master and a slave initiates upon receipt of the HPT DATA packet to the US1RX routine of the master BluReRun host.

By issuing the Create Connection command to the BT module the master should be able to establish a Bluetooth connection with any inquired device selected. After the connection has been completed, the BT module will return a Connection Complete event (Figure 5-). 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 HPT or the EXT device (through point to multipoint algorithm implementation) .

This time, correct function of the BluReRun was validated by establishing a connection between the master and the slave in a point to multipoint network. The layer 2 of the Parrotfish Project (by HPT BluReRun command interaction) issues the Inquiry command. In a point to multipoint network with the use of the point to multipoint algorithm the master HPT (and thus the master BluReRun) knows the handle and the name of the other HPTs or Parrotfish nodes in the vicinity. Therefore, all the necessary data transactions between the master and the slaves can be made.

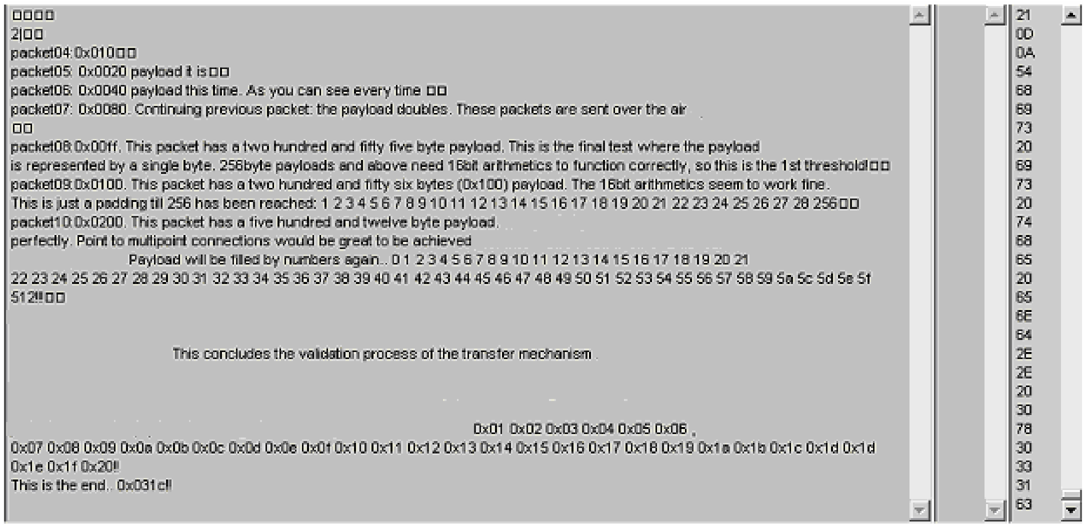
5.2.3.4 BluReRun nodes data exchange

In the requirements and the specifications set for the BluReRun, the system should be able to function as an efficient data "transporter" so that errorless and effective FPGA re/configuration could be achieved.

The BluReRun master or slave should have the ability, after they connect to each other, to exchange data that is received from the external device (HPT) in a full duplex manner. This data is in the form of ACL data packets and HPT data packets.

During the test processes in this stage, packets were formed and transmitted to the BluReRun slave or master by the HPT so that the BluReRun master or slave could forward these packets to the node indicated by the handle. The payloads of the packets ranged from 1 to 256 bytes, which is the maximum payload of the HPT data packet (in the form of bit streams for FPGA configuration). Validation of the BluReRun in this point meant that the HPT data packets would be transcribed correctly to ACL data packets, then they would be delivered correctly to the target BluReRun 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 HPT ones and transmit them correctly to the EXT DEV or the HPT in case we are referring to the Parrotfish architecture.

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 USART monitoring were used to validate that the HPT data packets are transcribed correctly to ACL ones and that the other end received the (correct) packets. In the place of EXT DEV were computers as well as three Parrotfish nodes.



```
□□□□
2|□□
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
□□
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.
perfectly. Point to multipoint connections would be great to be achieved
                        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 5c 5d 5e 5f
512!!□□


                                 This concludes the validation process of the transfer mechanism .



                                                                                     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 0x1d
0x1e 0x1f 0x20!!
This is the end.. 0x031c!!
```

```
21
0D
0A
54
68
69
73
20
69
73
20
74
68
65
20
65
6E
64
2E
2E
20
30
78
30
33
31
63
```

Figure 5-3: The master to slave and slave to master HPT data packet transfer

In the screenshot of Figure 5-3 12 packets that have been transmitted by the BluReRun master are shown, after they have been received by the slave. The slave transmits HPT data packets to the HPT or the external device, 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 master's data packet transmission mechanism and the slave's reception one were validated.

The same exactly screenshot describes the HPT data packet transfer from slave to master BluReRun node.

Efficiency

Since the BluReRun nodes have been able to exchange data the efficiency of the BluReRun protocol should be tested.

The data transfer mechanism efficiency is determined mainly by the data rate that it can achieve in both directions. BluReRun supports speeds of up to 460.8kbps. 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 (even in the HPT case of the external device the serial cable was used) is 128kbps and 56kbps is the most common one for dial-up modems.

The BluReRun protocol uses HPT data packets with a 3 byte header. In spite that it seems to achieve data rates of 57.24kbps with a 57.6kbps connection and 256bytes of data payload in the packet, it doesn't. This is because the Bluetooth protocol, which is involved in the BluReRun 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 56.00 kbps (7118.75bytes/sec) for Bluetooth ACL data packets with payloads of 256bytes. Thus, since the slower part of the protocol defines its speed, the data rate that BluReRun can achieve is 56.00kbps.

5.3 Parrotfish Project Experiments

IN HERE THE TWO BASIC EXPERIMENTS WHERE THE LAYERS INTERACTION WILL BE ADDED. I AM WAITING FOR THE SIMULATIONS FROM EFSTATHIOU….I AM GONNA WAIT FOR ANOTHER FOUR DAYS..AFTER THAT PROBLEMS WILL ARISE!!!

Conclusions and Future Work

Conclusions

A system capable of efficient and errorless wireless data transfer in an ad hoc environment for FPGA re/configuration on the boundaries of the Parrotfish project was introduced designed and implemented in the BluReRun thesis. The whole design has been embedded in a low-cost and low-power microcontroller that lacked large memory (1KB of SRAM) 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) 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 2 and can support every Bluetooth module that complies with that BT spec.

A Bluetooth device is usually able to find devices and accept connection requests from remote devices. The system implemented in this thesis adopted master to many slaves (point to multipoint) architecture. The slave BluReRun node 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 master BluReRun node retained the Bluetooth functions that enable it to find and identify Bluetooth devices. The characteristics of both the master and the slave BluReRun nodes have been integrated in every BluReRun node so that, when requested, each BluReRun node can act as a master or a slave. It also has the capability to establish a connection with one of the devices it found and finally exchange 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 errorless and efficient wireless data transfer for FPGA re/configuration is a reality. However, there were some certain issues that could be introduced as future work for the BluReRun system.

Future work

BluReRun is a complete data exchange system that can work on its own or as a subsystem (low layer) of the Parrotfish node. 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. 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 USART) for the connection between computers and BluReRun, because computers do not easily support higher USART speeds. The implementation of a USB core seems to be a viable solution. Though, the HPT does not support higher data rates.

II. Most Bluetooth profiles can be implemented through the BluReRun 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 use of a BluReRun HOST with more resources will boost the performance of the BluReRun protocol

IV. The integration of the three layers of the Parrotfish nodes in one PCB design will give a “black” box where everything from the top layer to the bottom one is embedded.

V. Most Bluetooth profiles can be implemented through the BluReRun 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)

Appendix A

In the APPENDIX A, four basic sections are examined:

a) The commands implemented and embedded in the BluReRun Host
b) The Glossary and terminology on Bluetooth and on the code used
c) The schematic and PCB of the BluReRun nodes

Commands Implemented

The Commands implemented and embedded in the BluReRun HOST are the following:

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
Switch Role	Used to switch role of the BT module from master to slave or vice versa
Read Local Name	Reads the user-friendly name of the local Bluetooth device
Reset	Resets the HC & the LM. The local BlueTooth device enters stand-by mode
Read Buffer Size	Reads the max size of the ACL & SCO packet payload that the HC can receive from the HOST
Set Event Filter	Specifies different event filters (the Host receives only events that interest it)
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
Change Local Name	Gives a user-friendly name to the local Bluetooth device
Remote Name Request	Obtains the user-friendly name of a remote device based on its BD_ADDR
Read BD ADDR	Reads the BD ADDR of the local device

Every Bluetooth command is specified in Bluetooth protocol version 2.0 [3].
Before moving on to the Bluetooth terminology it is vital for the

Bluetooth Acronyms and Terminology

A

ACL: **A**synchronous **C**onnection-**L**ess

ACL-C: **ACL-C**ontrol

ALU: **A**rithmetic **L**ogic **U**nit

API: **A**pplication **P**rogram(ming) **I**nterface

B

BD ADDR: **B**lueTooth **D**evice **Addr**ess

BlueApplE: **Blue**Tooth **Appl**ications **E**nvironment

BlueBridge: **Blue**Tooth Data **Bridge**

BluReRun: **Blu**etooth **Re**configurable **Run** Time Environment

BT module: **B**lue**T**ooth **module**

BTspec: **B**lue**T**ooth **spec**ification

BufLen: **Buf**fer **Len**gth (BluReRun flag indicating the number of stored bytes in the buffer)

BUFOVF: **Buf**fer **Ov**er**f**low (BluReRun flag indicating that the buffer is full)

C

CAGR: **C**ompound **A**nnual **G**rowth **R**ate

CID: **C**hannel **Id**entity

CLK: **Cl**oc**k**

CMOS: **C**omplementary **M**etal **O**xide **S**emiconductor

COM: **Com**munication

CPU: **C**entral **P**rocessing **U**nit

CRC: **C**yclic **R**edundancy **C**heck

D

DH: **D**ata - **H**igh Rate

DM: **D**ata - **M**edium Rate

E

ECED: **E**lectronics & **C**omputer **E**ngineering **D**epartment

EDR: **E**nhanced **D**ata **R**ate

EEPROM: **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory

EI: **E**nable **I**nput

EXT DEV: **Ext**ernal **Dev**ice

F

FEC: **F**orward **E**rror **C**orrection

FHSS: **F**requency **H**opping **S**pread **S**pectrum

FIFO: **F**irst **I**n, **F**irst **O**ut

FLAGREG: **Flag Reg**ister (BluReRun register containing various flags)

FPGA: **F**ield-**P**rogrammable **G**ate **A**rray

FTP: **F**ile **T**ransfer **P**rofile

G

GFSK: **G**aussian **F**requency-**S**hift **K**eying

GND: **G**rou**nd**

GOEP: **G**eneric **O**bject **E**xchange **P**rofile

GS: **G**roup **S**ignal

H

HCI: **H**ost **C**ontroller **I**nterface

HPT: **H**ardware **P**rogrammer & **T**ester

I

I/F: **I**nter**f**ace

I/O: **I**nput/**O**utput

IC: **I**ntergraded **C**ircuit

IEEE: **I**nstitute of **E**lectrical & **E**lectronic **E**ngineers

IEEE-SA: **IEEE** - **S**tandards **A**ssociation

INT: **Int**errupt

IP: **I**nternet **P**rotocol

IR: **I**nfra**R**ed

IrDA: **I**nfra**r**ed **D**ata **A**ssociation

ISM: **I**ndustrial, **S**cientific & **M**edical

ISR: **I**nterrupt **S**ervice **R**outine

L

L2CAP: **L**ogical **L**ink **C**ontrol & **A**daptation **P**rotocol

LAN: **L**ocal **A**rea **N**etwork

LAP: **L**AN **A**ccess **P**rofile

LED: **L**ight **E**mitting **D**iode

LM: **L**ink **M**anager

LMP: **L**ink **M**anager **P**rotocol

M

MAC: **M**edia **A**ccess **C**ontrol

µC: **m**icro**c**ontroller

MHL: **M**icroprocessor & **H**ardware **L**aboratory

O

OBEX: **Ob**ject **Ex**change Protocol

OCF: **O**pcode **C**ommand **F**ield

OGF: **O**pcode **G**roup **F**ield

Opcode: **Op**eration **code**

OPP: **O**bject **P**ush **P**rofile

P

PAN: **P**ersonal **A**rea **N**etwork

PCM: **P**ulse **C**ode **M**odulation

PDA: **P**ersonal **D**igital **A**ssistant

Q

QoS: **Q**uality **o**f **S**ervice

R

RAM: **R**andom **A**ccess **M**emory

RdBuf: **R**ea**d** **Buf**fer (BluReRun flag indicating where the next byte will be read from)

RF: **R**adio **F**requency

RISC: **R**educed **I**nstruction **S**et **C**omputer

RS232: **R**ecommended **S**tandard **232**

RX: **R**eceive

RXD: **R**eceived **D**ata

S

SCO: **S**ynchronous **C**onnection-**O**riented

SD: **S**ecure **D**igital

SIG: **S**pecial **I**nterest **G**roup

SP: **S**ynchronization **P**rofile

SPI: **S**erial **P**eripheral **I**nterface

SPP: **S**erial **P**ort **P**rofile

SRAM: **S**tatic **RAM**

T

TDD: **T**ime **D**ivision **D**uplex

TTL: **T**ransistor-**T**ransistor **L**ogic

TUC: **T**echnical **U**niversity of **C**rete

TX: **T**ransmit

TXD: **T**ransmit **D**ata

U

USART: **U**niversal **A**synchronous **S**yncronous **R**eceiver-**T**ransmitter (also serves as a BluReRun flag indicating that the USART 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

USxRX: **US**ART **x** (x can be 0 or 1) **R**eceive Complete ISRs in BluReRun

USxTX: **US**ART **x** (x can be 0 or 1) **T**ransmit Complete ISRs in BluReRun

W

WiFi: **Wi**reless **Fi**delity

WLAN: **W**ireless **LAN**

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

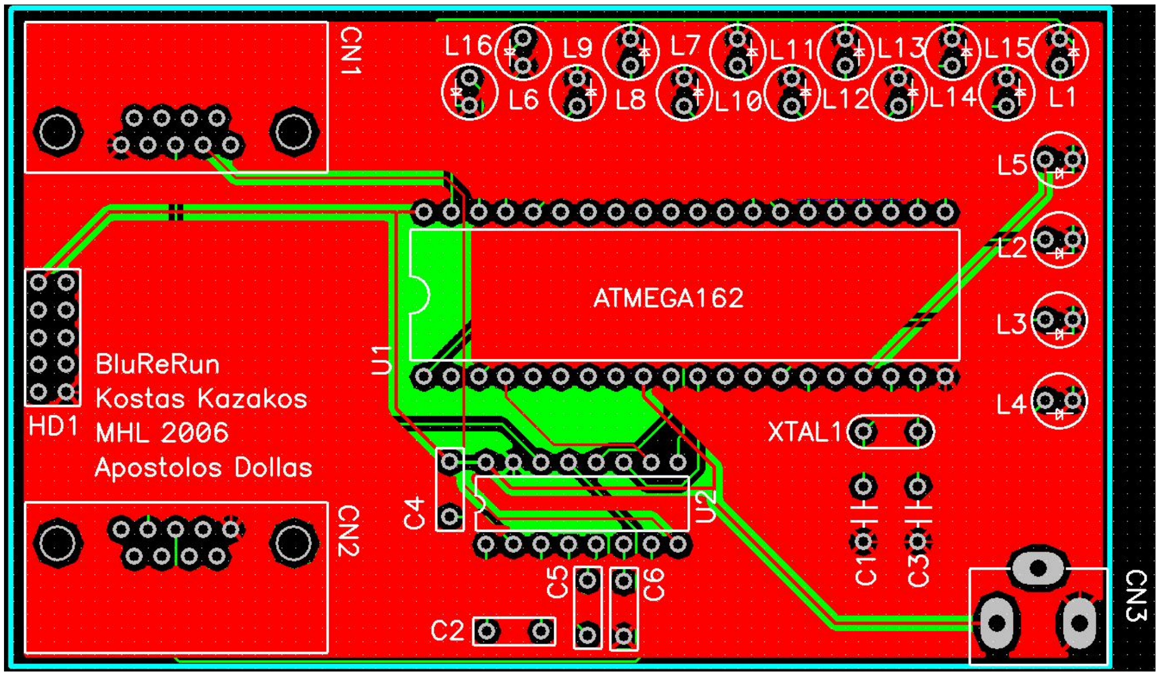
WrBuf: **Wr**ite **Buf**fer (flag indicating the buffer location to store the next byte)

X

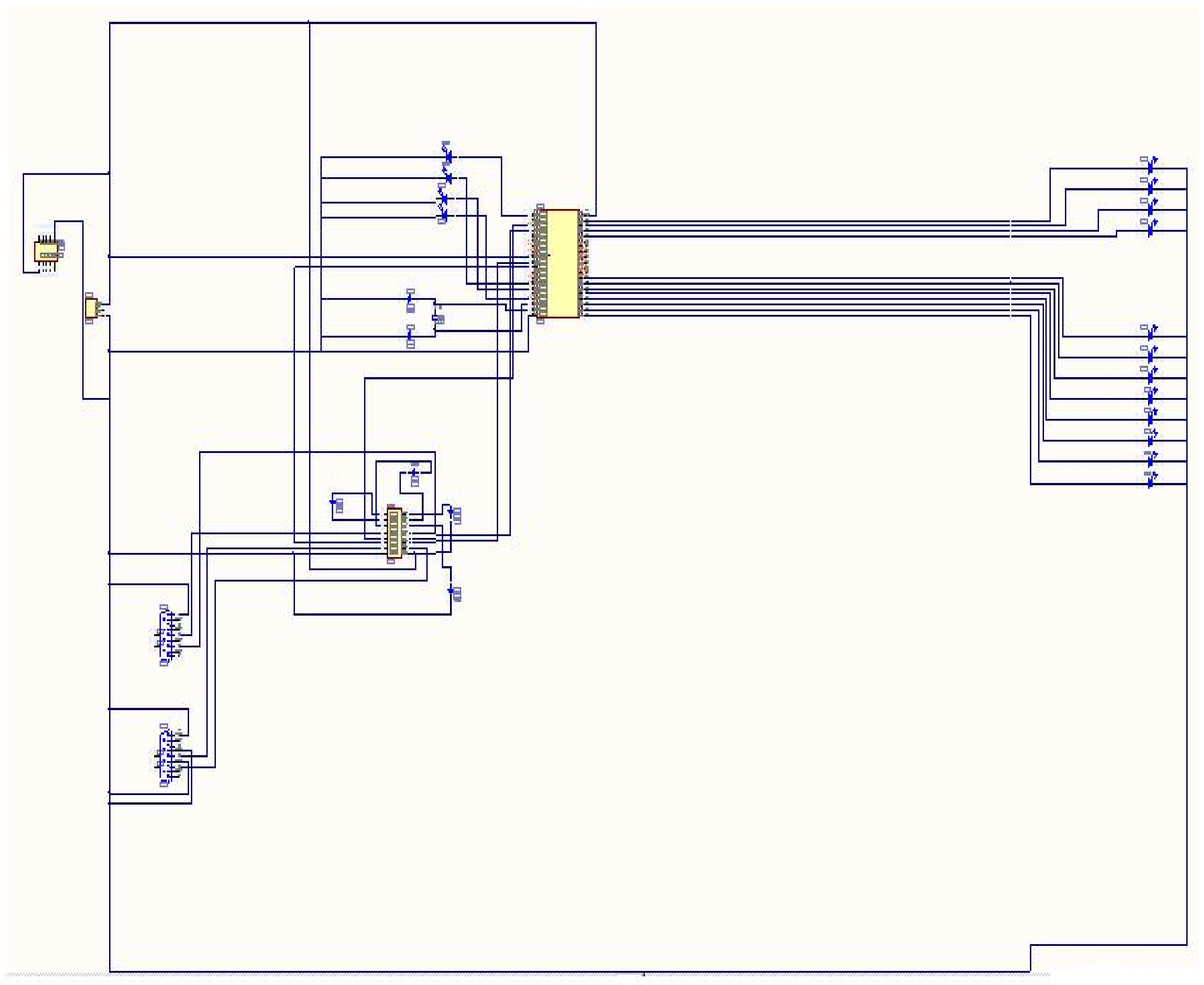
XTAL: Crystal

Schematic and PCB of BluReRun nodes

The PCB of the BluReRun nodes follows:



The Schematic of all three BluReRun nodes is the same. It is as depicted in the following picture:



Appendix B

ATmega162 AVR

The microcontroller (μC) used as the HOST (controls all the hardware components and synchronizes their operation) in BluReRun is the ATmega162 AVR by Atmel. The ATmega162 is pin compatible with ATMEGA161, which was used in the BluMiu architecture, thing that helped the transition from that architecture to BluReRun by making the latter a direct evolution of the packet mechanism and the transparency. The ATmega162 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 131 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 µC is 8MHz. Although an XTAL with a frequency of 11.0592MHz is used in BluReRun. This frequency has certain advantages over all the others: (1) It is one of the highest permitted by the ATMEGA162's datasheet (providing a higher performance) and (2) by using that frequency a 0% error rate can be achieved with the microcontroller's UART. [11]. he ATmega162 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
- 4 flexible Timer/Counters with separate Prescalers and Compare modes (2x8bit and 2x16bit)

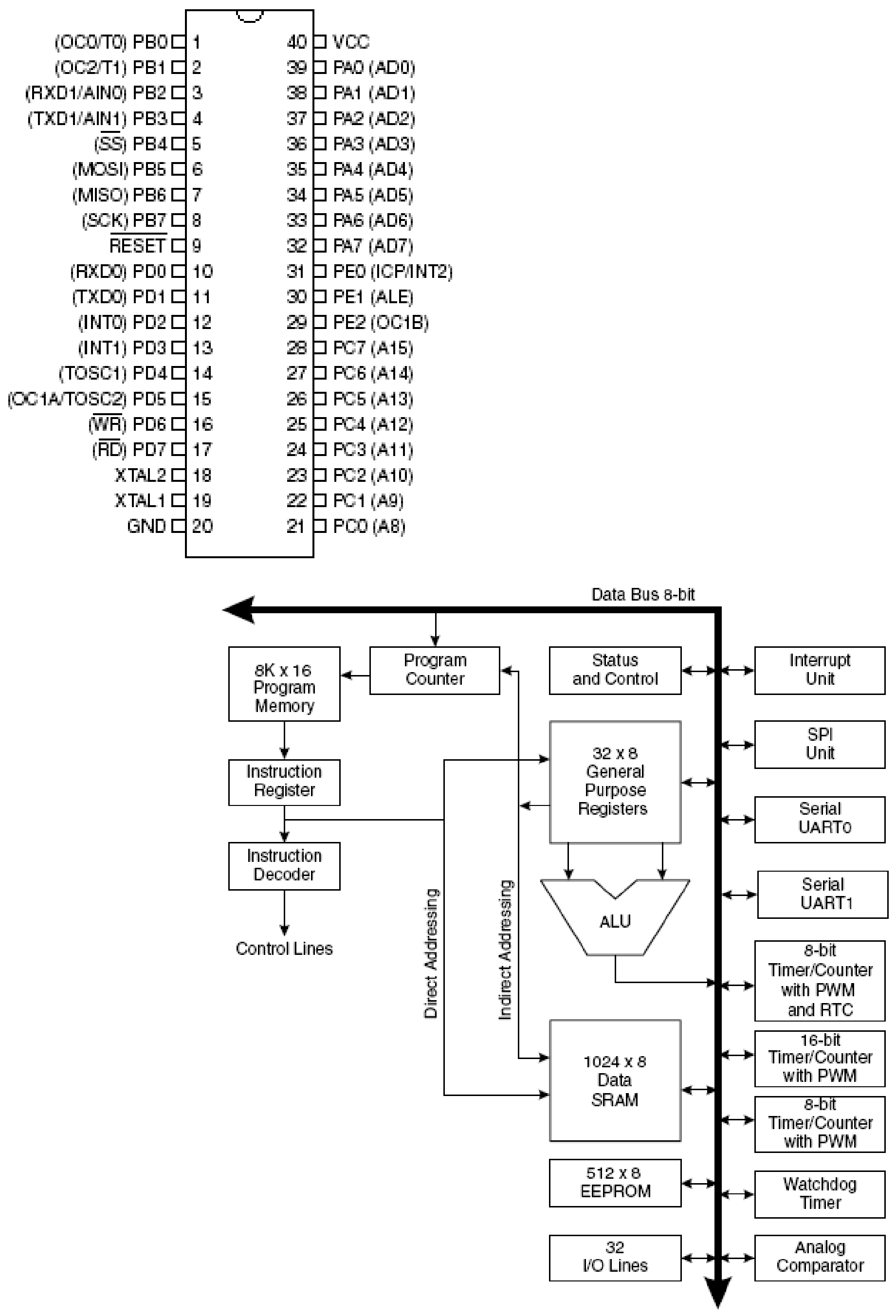


Figure 0-1 ATmega162 PIN configuration and architecture

- Real Time Counter with separate Oscillator
- 18 internal and 3 external interrupts
- 2 programmable serial USARTs
- 1 programmable Watchdog Timer with separate On-Chip Oscillator
- 1 SPI serial port

- 1 2-cycle multiplier
- 5 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

Also there are the Standby and Extensive Standby modes which are not used in the BluReRun design.

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.

Microcontroller resources used by BluReRun:

- ✓ 10.6KB of the Flash for the BluReRun Host
- ✓ (671+n*10)Bytes and (571+n*10)Bytes of the 1KB SRAM for the client and server respectively (n*10Bytes 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 HOST for the master-slave
- ✓ 2 of the 2 8 bit timers
- ✓ 0 of the 3 external interrupts
- ✓ Both USARTs
- ✓ Idle power saving mode is used

<u>USART</u>

The UART in ATmega161 has been replaced by a USART in ATmega162. The ATmega162 USART is compatible with the ATmega161 UART with one exception: The two-level Receive Register acts as a FIFO. The FIFO is disabled when the M161C Fuse is programmed. Still the following must be kept in mind when the M161C Fuse is programmed:
• The UDR must only be read once for each incoming data.
• The Error Flags (FE and DOR) and the ninth data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise, the error status will be lost. ATmega161 contains the baud-rate high-bytes for both UARTs in a common register – UBRRHI. ATmega162 has separate registers for the high-bytes of the two USARTs; UBRR0H and UBRR1H, implying a modification to the code when porting the design to ATmega162. Another minor difference is the initial value of RXB8, which is “1” in the UART in ATmega161 and “0” in the USART in ATmega162.

**AVR USART vs. AVR UART –
Compatibility**

The USART is fully compatible with the AVR UART regarding:

• Bit locations inside all USART Registers
• Baud Rate Generation
• Transmitter Operation
• Transmit Buffer Functionality
• Receiver Operation

However, the receive buffering has two improvements that will affect the compatibility in some special cases:
• A second Buffer Register has been added. The two buffer registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the ninth data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.
• The Receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DOR) error conditions.

The following control bits have changed name, but have same functionality and register location:
• CHR9 is changed to UCSZ2.
• OR is changed to DOR.

Appendix C

The Bluetooth module

The Bluetooth modules used are a kind donation of the Teleca Comtec Company to the Microprocessor and Hardware Laboratory. 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 007 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 1.2 as well as 2. [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 BluReRun design (as was done in the BluMiu) 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 [11]

A description of the ROK 101 007 and its functions will follow.

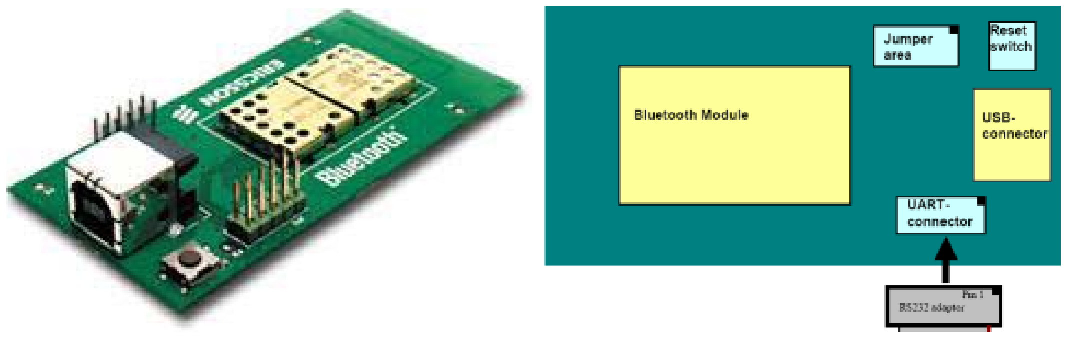


Figure 0-1 LZT 107 4123 R2A

ROK 101 007 is a short-range module for implementing BlueTooth functionality into various electronic devices. It is a type of Bluetooth module that supports the point to multipoint transfer of voice and data thus allowing the full function of the

BluReRun system and the form of Scatternets and piconets. It is compliant with BT spec 1.2, 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 2.3. 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. [15] 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.

References

Internet Resources

[2] Bluetooth Special Interest Group 2006, http://www.bluetooth.com/Bluetooth/SIG/

[6] The official Bluetooth Membership site, www.bluetooth.org

Literature

[1] *Bluetooth 2004: Poised for the mainstream*, In-Stat/MDR, 2004.

[3] *Specification of the Bluetooth System, Profile Book version 2*

[4] *Specification of the Bluetooth System, version 2*

[5] *Specification of the Bluetooth System, Covered Core Package Version 2*

[6] *Pradeep Puri, "The Parrot Fish community in Florida Everglades", MIT Marine Biology, MIT, 2005*

[7] 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.

[8] Elias Politarhos, Diploma Thesis, *"Wireless Communication and Applications of BlueTooth based on Microcontrollers"*, Technical University of Crete, Electronics & Computer Engineering Department, Microprocessor & Hardware Laboratory, 2004.

[9] 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.

[10] Alison Carter, *'Using Dynamically Reconfigurable Hardware in Real-Time Communication Systems"*, Literature Survey, University of York, 2005.

[11] *Atmel ATmega162(L) AVR datasheet*, Rev. 1228C-AVR-08/02, 2002 Atmel Corporation

[12] A.Dollas, D.Efstathiou, G.Vernardos, E. Polytarchos, K.Kazakos, *"On Distributed Reconfigurable Systems: Open Problems and Some Initial Solutions*", Proceedings of the Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), April 17 - 20, 2005, Napa, California, USA, (poster)

[13] *Atmel ATmega161(L) AVR datasheet*, Rev. 1228C-AVR-08/01, 2001 Atmel Corporation

[14] Getting Started - Bluetooth Application and Training Tool Kit, Ericsson Technology Licensing AB, 2005

[15] Ericsson ROK 101 007 Bluetooth PtmultiP Module datasheet

[16] D.Efstathiou, K.Kazakos, A.Dollas, *'Parrotfish: Task Distribution in a Low Cost Autonomous ad hoc Sensor Network through Dynamic Runtime Reconfiguration',* Proceedings of the Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), April 16 - 19, 2006, Napa, California, USA, (poster)