Hardware and software implications of creating Bluetooth Scatternet devices

David Johnson

Abstract—**This paper seeks to explain the practical issues encountered when implementing point to multipoint capable Bluetooth systems. System partitioning of the Bluetooth stack between the hardware components is a crucial component of any Bluetooth design and depends heavily on the bandwidth requirements of the product. The MIPS estimates of the various Bluetooth stack layers and some typical applications will be presented. Open source Bluetooth stacks running in open source embedded operating systems will be highlighted as a solution to building low cost Scatternet capable Bluetooth devices**

Index Terms—**Bluetooth, Open Source, Scatternet, System partitioning.**

I. Introduction

Since its conception in 1998, Bluetooth has promised to provide the world with a low power, short-range wireless link that will connect hundreds of devices existing in the personal area network space that surrounds us.

The Bluetooth (BT) system is described in the Bluetooth Specifications 1.1 [2] and supports a 1 Mbit/s gross rate in a so-called piconet, where up to 8 devices can simultaneously be interconnected. The radius of a piconet (transmission range—TR) is about 10 m for Class 3 devices and 100m for Class 1 devices. One of the key issues associated with the BT technology is the possibility of dynamically setting up and tearing down piconets. Devices or nodes can join and leave piconets as they move in and out of range. Different piconets can coexist by sharing the spectrum with different frequency hopping sequences, and interconnect in a scatternet. When all nodes are in radio visibility, the formation of overlapping piconets allows more than 8 nodes to simultaneously communicate and may enhance system capacity. In a multi-hop scenario, where nodes are not all in radio vicinity, a scatternet is mandatory to develop a connected platform for ad-hoc networking.

In reality creating scatternets between many portable devices such as mobile phones, Bluetooth headsets and PDA's is not quite as straightforward as it may seem. Bluetooth single chip IC's running at about 20 MIPS, commonly found in cell phones and headsets, promise point to multipoint functionality but unfortunately do not have enough processing power to maintain multiple connections by themselves. A second co-processor, which shares some of the Bluetooth stack load, is required on the device to make point to multipoint possible as is the case with a PC connected to a Bluetooth USB dongle.

D. Johnson is with the Information Society Technology centre at the Council for Scientific and Industrial research, Pretoria, South Africa (e-mail: djohnson@csir.co.za).

II. Bluetooth architecture

The hardware and software to support a Bluetooth link consists of several components as seen in Fig. 1.

The following hardware components are used [5]:

1) Optional Host controller: Computer to run the higher level code. It runs the Application software, upper layer of the Bluetooth protocol stack - profiles, logical link control and adaptation protocol (L2CAP), RFCOMM, and other stack functions above the HCI

2) Link control processor: A microprocessor that runs at least the lower layer of the stack. It may be combined with the host controller in embedded applications. It runs: Lower layer of the Bluetooth protocol stack - link manager protocol (LMP), containing the link manager (LM) and the link controller (LC). Software below the host control interface (HCI).

3) Baseband controller: Logic block to control the RF transceiver.

4) RF transceiver: Contains the RF synthesizer, VCO, mixers, Gaussian filter, clock recovery, and data detector.

5) RF front end: Contains the antenna bandpass filter, the transmit/receive switch and, if necessary, a low noise amplifier (LNA) and power amplifier (PA).

6) Antenna: May be internal or external, integrated on the PCB, or an OEM third-party component

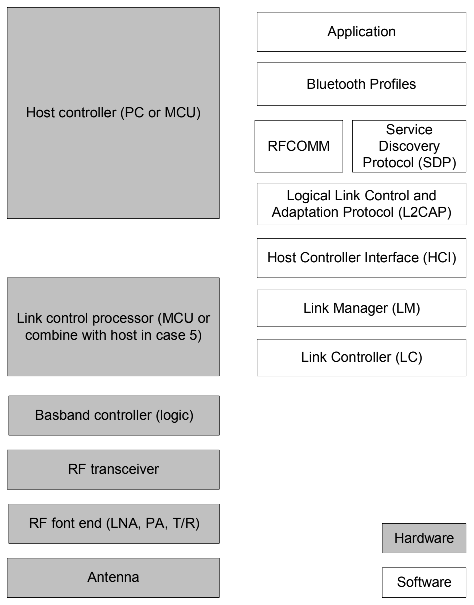
Host controller (PC or MCU)
Application
Bluetooth Profiles
RFCOMM
Service Discovery Protocol (SDP)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Link control processor (MCU or combine with host in case 5)
Link Manager (LM)
Link Controller (LC)
Basband controller (logic)
RF transceiver
RF font end (LNA, PA, T/R)
Antenna
Hardware
Software

Fig. 1. Components of a Bluetooth system

Most Bluetooth suppliers have adopted a multichip approach to system design, employing CMOS devices for the baseband core and microprocessor and bipolar devices for the RF functions (Fig. 2). While this approach helps simplify chip design, inherent disadvantages such as higher component count, inadequate board space, and other system integration issues, can lead to higher implementation costs. A typical implementation of a Bluetooth radio system, for example, involves a considerable number of relatively expensive RF and intermediate frequency (IF) filters. [1]

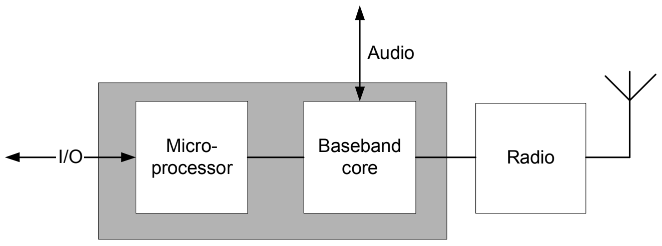

Fig. 2. Two IC Bluetooth solution

There are some Bluetooth suppliers that have achieved a single chip solution (Fig. 3). Here the microprocessor, baseband core and radio are all integrated onto one device entirely in CMOS [3]. The advantages of a single-chip solution are reduced cost, smaller footprint and a quick solution with little knowledge of design in RF circuitry required.

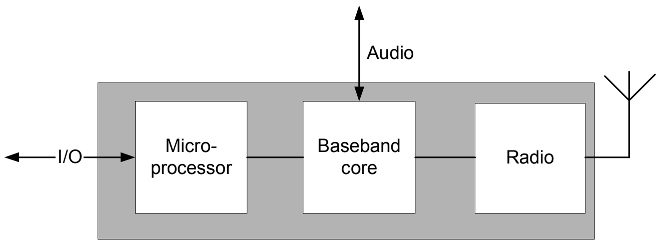

Fig. 3. Single IC Bluetooth solution

III. Bluetooth Stack Partitioning

Apart from the question of a single or dual IC solution for the Bluetooth module, there is also the question of how the stack is partitioned between the module microprocessor and the optional host controller.

With so many design options and architectural schemes on the market, designers are faced with a myriad of approaches to implementing Bluetooth technology in a system design. To choose the right option, designers must have a detailed understanding of the processing requirements of the application. They also need to understand the hardware and software approaches to implementing Bluetooth in system architecture. Fig. 4 also shows some of the possible cases that can be implemented in the system architecture [5].

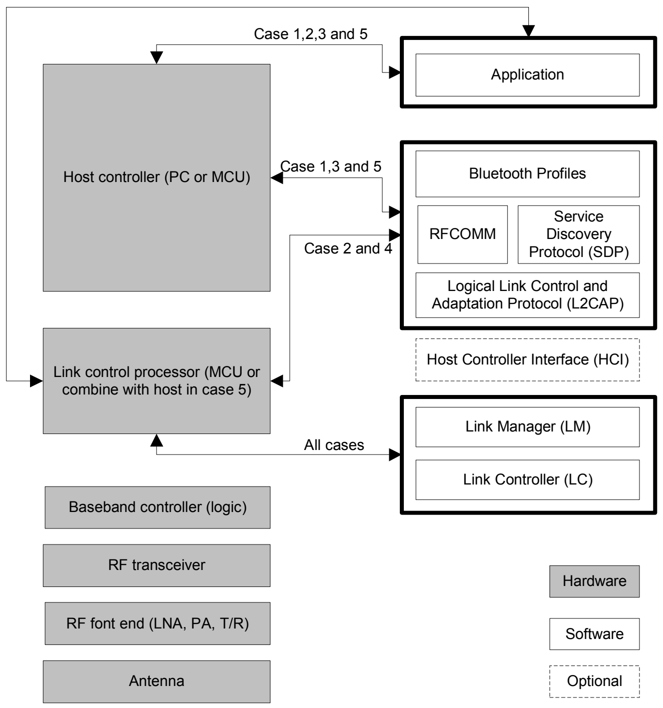
Case 1,2,3 and 5
Application
Host controller (PC or MCU)
Case 1,3 and 5
Bluetooth Profiles
RFCOMM
Service Discovery Protocol (SDP)
Case 2 and 4
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Link control processor (MCU or combine with host in case 5)
All cases
Link Manager (LM)
Link Controller (LC)
Baseband controller (logic)
RF transceiver
RF font end (LNA, PA, T/R)
Antenna
Hardware
Software
Optional

Fig. 4. Bluetooth stack partitioning cases

Case 1: PC-based design (*point-to-multipoint*). The host processor in this case is a PC running the product application code. It communicates through an API with the upper layer of the Bluetooth stack running on the PC. The stack software runs as a Bluetooth driver to the application and uses a serial interface to communicate with the lower layer of the stack running on dedicated hardware outside of the PC (for example a Bluetooth USB dongle). The serial interface uses HCI commands to control and exchange data and event timing with the lower layer stack.

Case 2: PC-based design 2 (*point-to-point*) An alternative implementation of case 1 is to use a single-chip solution, with the upper and lower layers of the stack integrated with the RF and baseband functions. In this case, the interface to the host processor is no longer HCI, but a chip-specific command set which handles the higher-level commands that are not part of the HCI.

Case 3: Stand-alone product – dual chip system (*point-to-multipoint*). In a stand-alone product, there is typically a microcontroller already included in the design. This microcontroller needs to be powerful enough to handle the full Bluetooth stack and the application. In addition to adequate processing power adequate memory resources are also important for point to multipoint functionality.

Case 4: stand-alone product – single chip solution (*point-to-point*). This architecture is the most cost effective of all five presented. It also represents the configuration targeted by most of the Bluetooth silicon vendors. The full Bluetooth stack and the application will run in the single chip. Many products that need basic point-to-point functionality, such as a Bluetooth headset, will use this architecture.

Case 5: Stand-alone product – ASIC (*point-to-multipoint*). If the designer has well-staffed hardware-description language (HDL) software teams available, and is simply looking for the lowest-cost solution for large production runs, then this last architecture may be the answer. It consists of an integrated baseband and microcontroller, with an external RF transceiver. In this scenario, a high-performance microcontroller handles all the software tasks, from application to upper and lower layers of the stack. In addition, baseband IP can be acquired and integrated on an ASIC, together with application circuits.

IV. MICROPROCESSOR CONSIDERATIONS

Bluetooth IC vendors embed small, low-power processors such as ARM7, Hitachi H8 or Motorolla Dragonball into their IC's, which have enough processing power to run the Bluetooth stack. The following are typical MIPS estimates for the upper and lower layers of the stack [3].

Baseband Layer (LC, LM and HCI): 8-12MIPS

Host Protocol Layer (HCI, L2CAP, RFCOMM, SDP):1-2 MIPS

The baseband layer is a real-time environment that has 625μs blocks to process received data, hence it's larger processing requirement. The application layer is very difficult to quantify as it largely depends on other protocols and services defined by the relevant profile. A headset, for example will impose a minor additional processor load, while a LAN access point with IP routing and multipoint functionality will exhibit significant additional processor loading. [3]

Single-IC vendors manage to create a full application such as a Bluetooth headset on one IC. These IC's have microprocessors running at about 20MIPS, which is just sufficient for the baseband layer, host protocol layer and the small application.

V. OPEN SOURCE SOLUTIONS FOR BLUETOOTH

Once the hardware solution has been chosen, the Bluetooth protocol layers and the application layer need to be loaded into non-volatile memory of the device.

If a point-to-point solution is needed, for example replacing a serial cable between two devices, a basic serial port profile can be loaded into a module on each end of the link and a point to point connection can be established. For example, the Cambridge Silicon Radio development kits allow the user to load point-to-point applications with the full Bluetooth stack into the flash memory of a Bluetooth module and place these in products.

However when an embedded point to multipoint solution is needed, as explained in the section on Bluetooth stack partitioning, an extra microprocessor is needed to run the upper layers of the Bluetooth stack and the application. An embedded operating system is also required on the microprocessor to handle memory, scheduling, and other system tasks. Commercial embedded operating systems such as VXWorks or Nucleus and Bluetooth stacks such as Mezoe or Windigo are available but at a substantial cost with royalty fees on each product produced.

There are a number of robust open source embedded operating systems that support microprocessors without memory management such as eCos and uClinux. Similarly there are now four open source Bluetooth stacks available for FreeBSD and Linux which can be ported to embedded environments [6]. These are the FreeBSD Bluetooth stack, OpenBT developed by Axis, Bluez which is part of the Linux kernel and Affix developed by Nokia.

An example of a commercial open source Bluetooth product is the Axis access point [4]

VI. Point to multipoint serial port adapter

The point-to-multipoint serial port adapter was created by the author at the CSIR in 2003 for a heart rate over Bluetooth system which monitors multiple athletes' heart rates from the edge of a sports field. It was used for the 20/20 Supersport cricket series held in South Africa during April 2004. The system shown in Fig. 5 sends heart rate information, which is measured using a sensor on the athletes' chest, to a heart rate slave module via magnetic induction. This information is sent from the heart rate slave module via Bluetooth to a connected class 1 master unit which can be up to 100m away. The master unit is pre-programmed to connect to a set of Bluetooth slave units and when it is connected to these slave units, it relays any information it received from any of the slaves via its RS232 or RS485 connection to a host PC. This PC will be running a heart rate monitor application which can view a multiple number of athlete's heart rates. The host PC does not have any local Bluetooth stack, it receives heart rate information via the serial port either using polling commands or using a multiplexing protocol.

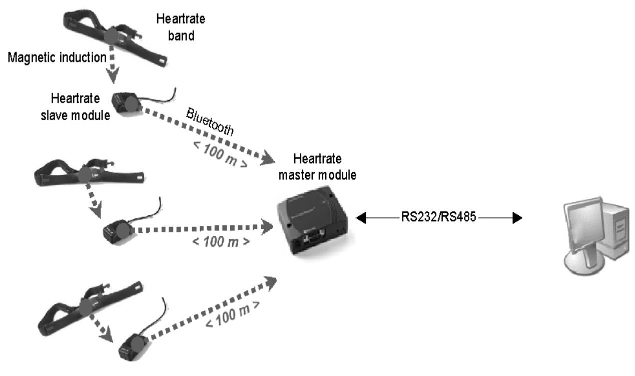

Fig. 5. Bluetooth heart rate monitoring system

The heart rate slave module uses the stack partitioning architecture outlined in case 4. As shown in Fig. 6 it runs the entire stack and application in a single microprocessor. The slave device was developed using the Cambridge Silicon Radio Bluelab development framework environment which creates a special user space called a virtual machine. This virtual machine ensures that the application cannot interfere with the real time requirements of the lower levels of the stack.

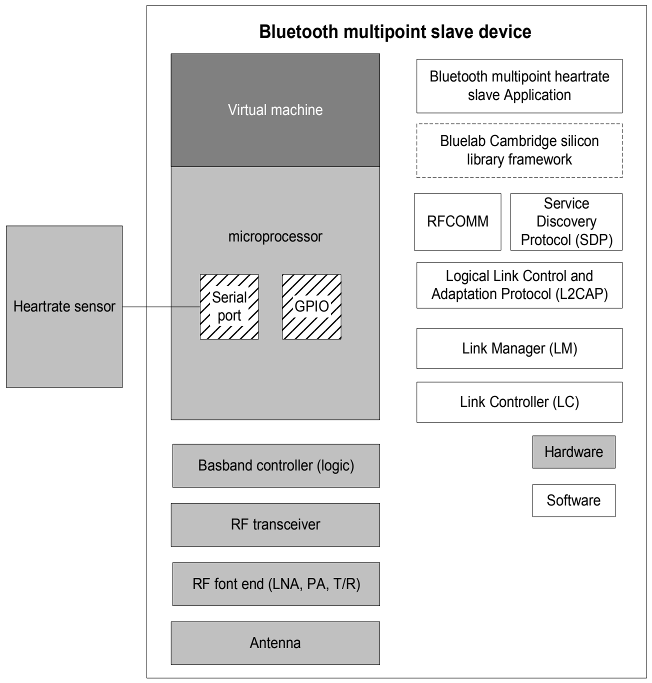
Bluetooth multipoint slave device
Virtual machine
microprocessor
Serial port
GPIO
Heartrate sensor
Basband controller (logic)
RF transceiver
RF font end (LNA, PA, T/R)
Antenna
Bluetooth multipoint heartrate slave Application
Bluelab Cambridge silicon library framework
RFCOMM
Service Discovery Protocol (SDP)
Logical Link Control and Adaptation Protocol (L2CAP)
Link Manager (LM)
Link Controller (LC)
Hardware
Software

Fig. 6. Bluetooth heart rate slave device system partitioning

The heart rate master module was created using uClinux and the Bluez stack running on an ARM based Atmel AT91 series microprocessor with no memory management, 4meg of RAM and 2Meg of FLASH. The architecture for this serial port adapter is shown in Fig. 7. This architecture uses the stack partitioning outlined in case 3.

The Bluetooth multipoint application is responsible for handling host to module communication (data and commands) and setting up Bluetooth links to the slave devices. Commands from the host to the application are carried out using AT commands over the serial interface. These AT commands can start an inquiry, list slave devices that the master should connect to, set up baud rates etc. It is also capable of sending data, which it received from the slave module, either using a polling method such as MODBUS or using a multiplexing protocol such as the one seen in Fig. 8.

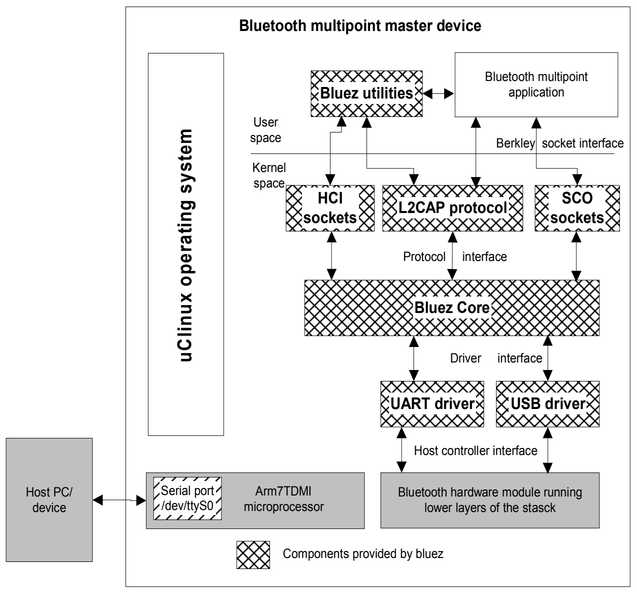

Fig. 7. Bluetooth master device system partitioning

The multiplexing protocol (Fig. 8) embeds the length of the frame, the Bluetooth address, command and data within the message. A Bluetooth address of 0x00 is used to broadcast a message to all the slaves.

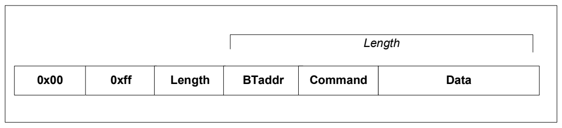

Fig. 8. Multiplexing protocol between host and Bluetooth master device

A polling method is useful in an industrial environment where Bluetooth is used to replace cables and a MODBUS protocol is already being used. The multiplexing protocol should be used where a new point to multipoint application is being developed and the maximum data rate of the combined slaves is less than the maximum data rate of the communication between the host and master unit.

VII. Conclusion

Creating low cost scatternet enabled Bluetooth devices cannot be achieved using a single chip solution. In a stand-alone product, an additional microprocessor is needed which is powerful enough to handle the full Bluetooth stack and the application which sets up multiple connections. In addition to adequate processing power, adequate memory resources are also important for point to multipoint functionality. The use of open source operating systems such as uClinux and open source Bluetooth stacks such as Bluez running on this microprocessor make it possible to build scatternet enabled Bluetooth devices which are affordable and easily customized to any required application. This Bluetooth device can multiplex information from multiple slaves to an external port connected to a host PC or other long range transmitting device such as a GSM modem using a multiplexing protocol or a polling method such as MODBUS.

References

[1] M. Phillips, “Reducing Bluetooth components”, Electronic Engineering Times, Issue 1134, Oct. 2000, pp. 88

[2] Specification of the Bluetooth System, Volume 1, Core, version 1.1, http://www.bluetooth.com, 22 Feb. 2001.

[3] J. Bray, C. F. Sturman, “Bluetooth 1.1: Connect without cables”, 2nd ed, Prentice Hall, 2001, pp. 480-492

[4] A. Karlsson, “Wireless open source platform”, M.S. Thesis, Dept. of Telecommunications and Signal Processing, Blekinge Institute of Technology, Dec. 2000

[5] S. Walton, R. Kumar, “Reducing time to market for Bluetooth-enabled products”, CommsDesign, Oct. 2003

[6] D. Xiaoyong Yang, “Bluetooth Enabled Embedded Linux”, Nanyang Technology University, Singapore, Linux Congress, 2002