BLUETOOTH FM AUDIO TRANSMITTER

Amit Das
Buck Golem
Siddharth Gopalan

Dec 12th, 2006

ECE 395: Advanced Digital Projects Lab
University of Illinois at Urbana-Champaign
Abstract
We attempted to build a Bluetooth Audio FM Transmitter that would stream mp3 music from a PC to our device, which will act as an FM transmitter to enable radios to receive the audio. We intended to use the A2DP (Advanced Audio Distribution Profile) profile in order to accomplish streaming via Bluetooth. A2DP uses the increased bandwidth of the new Bluetooth standard in order to stream high-quality audio. Due to technical difficulties with the audio streaming, we changed our design to an mp3 player that could be updated via Bluetooth.
# Table of Contents

Terms and Keywords........................................................................................................ 4  
Overview ........................................................................................................................ 5  
Stack ............................................................................................................................... 5  
Adding an SDP Record ................................................................................................. 7  
Advanced Audio Distribution Profile (A2DP) ............................................................. 11  
Bluetooth MP3 Player .................................................................................................. 13  
Playing Audio .............................................................................................................. 13  
dsPIC ........................................................................................................................... 14  
LCD .............................................................................................................................. 16  
DOSonChip .................................................................................................................. 16  
References ................................................................................................................. 18  
Communication buses ............................................................................................... 18  
Bluetooth ................................................................................................................... 18  
UART ............................................................................................................................ 18  
RS232 ........................................................................................................................ 18  
I2C ............................................................................................................................... 19  
SPI ............................................................................................................................... 19  
Subversion ................................................................................................................. 20  
Eagle ............................................................................................................................ 20  
PCB ............................................................................................................................ 21
Terms and Keywords

- ACL – Asynchronous Connection Less
- DAC – Digital to Analog Converter
- DRC – Design Rules Check
- HCI – Host Controller Interface
- I2C – Inter Integrated Circuit bus
- L2CAP – Logical Link Layer and Adaptation Protocol
- LMP – Link Manager Protocol
- PCB – Printed Circuit Board
- RFCOMM – Radio Frequency Communication (serial port emulator)
- SCO – Synchronous Connection Oriented
- SDP – Service Discovery Profile
- SPI – Serial Peripheral Interface
- SPP – Serial Port Profile
- SVN – Subversion
- UART – Universal Asynchronous Receiver Transmitter
- ID3 – Metatag data for the MP3
Bluetooth

Overview
Bluetooth is a short-range wireless technology that is currently used mostly to connect hands-free devices to mobile phones. There are two major versions of Bluetooth:

<table>
<thead>
<tr>
<th>Version</th>
<th>Maximum Transfer Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 – 1.2</td>
<td>700 kbps</td>
</tr>
<tr>
<td>2.0</td>
<td>3 Mbps</td>
</tr>
</tbody>
</table>

Stack
We got a lot of the details of the Bluetooth stack from a paper on BeremoTe: A Bluetooth Project by Steffen Larsen and Jesper Hannson. The Bluetooth protocol is divided into different layers to make it modular and provide a nicer interface to the application developers. The following block diagram depicts the higher level stack organization of a Bluetooth profile. This is an example of the stack implemented for the use of the Generic Audio Video Distribution Profile (GAVDP).

![Figure 1: Upper Layers of Bluetooth Stack](image)

Figure 1: Upper Layers of Bluetooth Stack
The following block diagram is an overview of the lower level stack of the Bluetooth device. This part of the stack mainly deals with hardware control and interface.

**Figure 2: Lower Layer of the Bluetooth Stack**

**Bluetooth Radio**
Bluetooth transmits data over the 2.4GHz spectrum. The Bluetooth radio utilizes the concepts of piconets and frequency hoping. This layer implements the actual radio link and controls the physical sending and receiving of radio signals.

**Baseband**
This layer sends and receives packets over the RF layer. It is also responsible for flow control and synchronization of clocks. There are two types of links that can be established - Synchronous Connection Oriented (SCO) and Asynchronous Connection Less (ACL). SCO connections are synchronized (as the name suggests) and have low bandwidths of about 64kb/s. ACL connections are packet-based and have much higher bandwidth.
Link Manager Protocol (LMP)
This layer is responsible for setting up the SCO and ACL connections. Applications can access the LMP through the Host Controller Interface (HCI).

Logical Link Control and Adaptation Protocol (L2CAP)
This layer offers both SCO and ACL layers to the upper protocol layers. All the asynchronous data is handled by this layer. The L2CAP manages links through different Protocol and Service Multiplexers (PSMs).

Radio Frequency Communication (RFCOMM)
This runs on top of the L2CAP layer and is used to emulate a serial port. The protocol is a reliable connection-oriented streaming protocol. The RFCOMM ports are mainly used for establish a serial link between two devices. The serial links are low-bandwidth connections and hence cannot be used for streaming audio.

Service Discovery Protocol (SDP)
This protocol informs other devices the services that this device offers. For instance, a Bluetooth headset would inform other devices that it offers a headset service. It provides the application capabilities of the device. A service discovery is an absolute prerequisite before setting up any kind of data transfer connection between two Bluetooth devices.

Adding an SDP Record
An SDP record is needed for each of the services offered by a Bluetooth device. For instance, a device would need separate SDP records for the Serial Port profile and the Dial-Up Networking profile. We needed to add an SDP record for the A2DP service on the chip. Unfortunately, the details on creating SDP records are very vague, so we had to piece it together from a combination of code snippets and specs.

The SDP record uses a little-endian format. The different types of entries are:

Single Data Entry
Attribute ID (2 bytes), Type and Size (1 byte), Data

Data Sequence Entry
Attribute ID (2 bytes), Data Sequence ('35'), Size of Data Sequence (1 byte), Type & Size of first entry (), Data, Type & Size of second entry, Data ...

DUN SDP Record Example
00,00,0A,00,00,00,01,00,35,03,19,03,11,04,00,35,0C,35,03,19,00,01,35,05,19,03,00,8,02,05,00,35,03,19,02,10,06,00,35,09,09,6E,65,09,6A,00,09,00,01,09,00,35,08,35,06,19,03,11,09,00,01,00,05,13,44,69,61,6C,2D,75,70,20,4E,65,74,77,6F,72,6B,69,6E,67,00,05,03,28,00
<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,00</td>
<td>Attribute ID = Record Handle</td>
</tr>
<tr>
<td>0A</td>
<td>Type &amp; Size</td>
</tr>
<tr>
<td>00,00,00</td>
<td>Data = First entry in the SDP Service Table</td>
</tr>
<tr>
<td>01,00</td>
<td>Attribute ID = Service Class ID</td>
</tr>
<tr>
<td>35</td>
<td>Start of Data Sequence</td>
</tr>
<tr>
<td>03</td>
<td>Size of Data Sequence = 3 Bytes</td>
</tr>
<tr>
<td>19</td>
<td>Type &amp; Size of first entry = UUID, 2 Bytes</td>
</tr>
<tr>
<td>03,11</td>
<td>Data = DUN</td>
</tr>
<tr>
<td>04,00</td>
<td>Attribute ID = Protocol Descriptor List</td>
</tr>
<tr>
<td>35</td>
<td>Start of Data Sequence</td>
</tr>
<tr>
<td>0C</td>
<td>Size of Data Sequence = 12 Bytes</td>
</tr>
<tr>
<td>35</td>
<td>Data Sequence 1 (Element sizes are different)</td>
</tr>
<tr>
<td>03</td>
<td>Size of Data Sequence 1 = 3 Bytes</td>
</tr>
<tr>
<td>19</td>
<td>Type &amp; Size of first entry = UUID, 2 Bytes</td>
</tr>
<tr>
<td>00,01</td>
<td>Data = L2CAP</td>
</tr>
<tr>
<td>35</td>
<td>Data Sequence 2</td>
</tr>
<tr>
<td>05</td>
<td>Size of Data Sequence 2 = 5 Bytes</td>
</tr>
<tr>
<td>19</td>
<td>Type and Size of first entry = UUID, 2 Bytes</td>
</tr>
<tr>
<td>03,00</td>
<td>Data = RFCOMM</td>
</tr>
<tr>
<td>08</td>
<td>Type and Size = UINT, 1 Byte</td>
</tr>
<tr>
<td>02</td>
<td>Channel 2</td>
</tr>
<tr>
<td>05,00</td>
<td>Attribute ID = Browse Group List</td>
</tr>
<tr>
<td>35</td>
<td>Start of Data Sequence</td>
</tr>
<tr>
<td>03</td>
<td>Size of Data Sequence = 3 Bytes</td>
</tr>
<tr>
<td>19</td>
<td>Size &amp; Type of first element = UUID, 2 Bytes</td>
</tr>
<tr>
<td>02,10</td>
<td>Data = Public Browse Group</td>
</tr>
<tr>
<td>06,00</td>
<td>Attribute ID = Language Base</td>
</tr>
<tr>
<td>35</td>
<td>Data Sequence</td>
</tr>
<tr>
<td>09</td>
<td>Size of Data Sequence = 9 Bytes</td>
</tr>
<tr>
<td>09</td>
<td>Size &amp; Type of first element = UINT, 2 Bytes</td>
</tr>
<tr>
<td>6E,65</td>
<td>Data (Some language - dont know which one)</td>
</tr>
<tr>
<td>09</td>
<td>Size &amp; Type of first element = UINT, 2 Bytes</td>
</tr>
<tr>
<td>6A,00</td>
<td>Data</td>
</tr>
<tr>
<td>09</td>
<td>Size &amp; Type of first element = UINT, 2 Bytes</td>
</tr>
<tr>
<td>00,01</td>
<td>Data</td>
</tr>
<tr>
<td>09,00</td>
<td>Attribute ID = Profile Descriptor List</td>
</tr>
<tr>
<td>35</td>
<td>Data Sequence</td>
</tr>
<tr>
<td>08</td>
<td>Size of Data Sequence = 8 Bytes</td>
</tr>
<tr>
<td>35</td>
<td>Data Sequence 1 (Element sizes are different)</td>
</tr>
<tr>
<td>06</td>
<td>Size of Data Sequence 1 = 6 Bytes</td>
</tr>
<tr>
<td>19</td>
<td>Type &amp; Size of first entry = UUID, 2 Bytes</td>
</tr>
<tr>
<td>03,11</td>
<td>Data = DUN</td>
</tr>
<tr>
<td>09</td>
<td>Type &amp; Size = UINT, 2 Bytes</td>
</tr>
<tr>
<td>00,01</td>
<td>Data = Version 0100</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>00,01</td>
<td>Attribute ID = Service Name</td>
</tr>
<tr>
<td>25</td>
<td>Size &amp; Type of data = String with size contained in next 8 bits</td>
</tr>
<tr>
<td>13</td>
<td>Size of data = 19 Bytes</td>
</tr>
<tr>
<td>44,69,-67,00</td>
<td>Data = ASCII Values for Service Name</td>
</tr>
<tr>
<td>05,03,28,00</td>
<td>Don't know what this is.</td>
</tr>
</tbody>
</table>

The A2DP SDP record we created was:
00,00,0A,00,00,00,00,01,00,35,03,19,0B,11,04,00,35,10,35,06,19,00,01,09,19,00,35,06,19,19,00,09,00,01,05,00,35,03,19,02,10,06,00,35,09,09,6E,65,09,6A,00,09,00,01,09,00,35,08,35,06,19,0D,11,09,00,01,00,01,25,0D,"A2DP HEADSET",00
**Host Controller Interface (HCI)**

The Host Controller Interface provides interfacing methods to access the Bluetooth hardware. The following figure provides an overview of the lower software layers of the stack. The HCI implements all commands for the Bluetooth hardware and the higher stacks by accessing the baseband radio, link manager, hardware status registers, control registers and event registers. It handles all communication links between Bluetooth devices through the Link Manager and by controlling the Bluetooth baseband radio.

![HCI Layer Diagram](image)

There can be multiple layers between the higher level stacks and the hardware. They are the HCI transport layers; these layers provide a physical connection (bus) to transfer data between the Host and the controller interface. Asynchronous alerts of HCI events are sent to the Host via communication buses like UART and USB. The LMX9830 Bluetooth chip used UART protocol to carry out ACL/ HCI event notifications. Commands from the host and other higher applications layers such as A2DP send commands such as Inquire, Get Capabilities to the HCI which are then interpreted as opening wireless ports and setting up ACL or SCO connections. These ports can then be used to swap protocol requests and indications to initialize the Bluetooth devices and prepare them for more specific data transfer connections such as streaming audio, streaming video, dialup connections etc.
Advanced Audio Distribution Profile (A2DP)
This profile allows high-quality stereo sound to be streamed from one device to another over Bluetooth. The protocol needs the higher bandwidth offered by Bluetooth v2.0 and hence has become popular only recently. Most systems still do not support the A2DP profile. Although A2DP is not supported by Windows XP, many Bluetooth dongles come with their own stack that supports A2DP. The A2DP profile relies on the AVDTP protocol (Audio and Video Distribution Transmission Protocol). The AVDTP protocol provides standards for initiating streaming connections between two devices. To initiate a streaming connection, an SCO link has to be first created. After the SCO link is created, the faster ACL link is created between the two devices in order to stream the actual music.

The LMX9830 chip did not have any of the upper-layer protocols in its stack. Hence, we had to implement both the AVDTP protocol and the A2DP profile in order to stream music. After a lot of searching, we realized that the LMX9830 chip does not provide applications access to the L2CAP layer. We needed to access the L2CAP layer in order to initiate ACL connections. The chip only provided access to the RFCOMM layer that was used for serial port SCO connections. Since we could not implement the AVDTP protocol, we were unable to stream music.

Reference Documents
- Specifications for all the Bluetooth profiles
- LMX9830 Datasheets and Specifications

Bluetooth Hardware
Possibly the greatest hurdle of our project was finding the proper Bluetooth hardware. Specifications that seemed simple in the beginning turned out to not be so straightforward. In essence, we wanted a chip that was affordable to buy, had a straightforward and inexpensive development path, and detailed documentation. These are the basic requirements for buying any piece of hardware, but proved near impossible to find in the area of Bluetooth. We also wanted a chip that could handle Enhanced Data Rate (EDR) bandwidth and that provided the necessary functionality upon which to build A2DP. This is the process that we went through:

The first Bluetooth chip that we considered seemed nearly perfect. The chip was the BCM2037 from Broadcom. This chip featured built-in support for EDR, A2DP and the remote control profile (AVRCP). After several days of searching on where and how to obtain this part, we determined that the chip was quite new and at this point was only available at the high-volume corporate level. This turned out to be the case for most Bluetooth chips, especially chips with advanced features built in.

The second chip that we looked at had comparable features. It was the BlueCore4 audio chip from CSR. This was not their newest model, and was available at a reasonable price. The chip had hardware support for EDR and source code for A2DP is provided with the development kit. After looking at how our project would be developed using
this chip, we found that the only choice was to buy CSR’s custom built development studio, which costs $3000. This was obviously not an option.

Our third try was the LMX9830 chip from National Semiconductor. The specification sheet was quite large, and showed that many of the needed lower layers were present in the chip. After further inspection, we noticed that key information was left out of the spec sheet. The opcodes, which determine which function is being requested by the microcontroller, were not in the spec sheet. After some search, we found that the opcodes were in the documentation that came with the $250 development kit. At this point, it had been several weeks with no progress on the project, and we saw little prospect of a better alternative; so we bought the development kit and started development.

The software user’s guide that came with the development kit was over two thousand pages long, and was quite thorough. We had success in adding the A2DP service record to the chip so that the chip was recognized as an A2DP device, but the A2DP connection failed. This was expected, since no procedure had been implemented to complete the A2DP connection. Several weeks went by while getting the microcontroller up to the level needed to implement this procedure. When it came time to finally implement the A2DP connection, we examined the A2DP documents and determined that certain L2CAP connections needed to be established in a specific order. After searching through the documentation, we found that there was no direct access to the L2CAP layer – only to the higher level and very specific purpose RFCOMM layer. This made it impossible to implement A2DP with the hardware.

After looking at the options for some time more, we have come to the conclusion that the best approach to the completion of this project would be to use a Bluetooth chip which only implements the lowest levels of the stack (up to the Host-Controller Interface) and implement the rest of the project in software either on a microcontroller or a full-fledged microprocessor. BTnodes is an open-source project which may provide the software basis for this (http://www.btnode.ethz.ch/).
Bluetooth MP3 Player
When we found that the LMX9830 did not provide access to the L2CAP layer/Host Controller Interface (HCI) we realized we would not be able to stream the audio over Bluetooth. We changed our design to that of an mp3 player that could be updated via Bluetooth. The overview of our system is given below:

In the new design, the Bluetooth link from the PC to the LMX9830 chip function as a serial cable replacement. We used the Serial Port Profile in the chip to receive complete mp3 files from the PC. The LMX9830 chip was connected to the PIC microcontroller via a UART interface. The microcontroller coordinated the activities of the other chips. The remaining system can be divided into two main parts – one part converted the mp3 file to an audio output while the other stored the mp3 files on a SD card.

Playing Audio
The PIC was connected to the mp3 chip via an I2C and a SPI bus. The mp3 chip decoded the music and sent it to a DAC (Digital to Analog Converter). The DAC outputted an analog signal that was amplified and sent to the speakers. We decided not to use the configuration lines in the DAC; instead we used the default settings for the DAC and configured the mp3 chip to work with those settings.
The dsPIC30F4013 was used as the microcontroller for our circuit. It was one of the few chips which included a C Compiler for coding. It was also a chip which had all the necessary communication bus protocols needed for our circuit. It is well documented and provides access to a variety of development resources.

The dsPIC was implemented as the transport interface between all the peripheral devices. Most chips used different communication protocols; hence all data had to be routed to each peripheral via the PIC microcontroller. The PIC microcontroller became the host for the Bluetooth chip performed the 2007 register initializations required for the functioning of the MP3 decoder. Three separate communication protocols were used – I2C, SPI and UART.

PIC coding and debugging was carried out with the help of the C30 Compiler, the MPLAB Instructional Development Environment and the In circuit debugger. These were very useful especially while implementing the UART protocol.
Coding the UART protocol for the PIC was pretty tricky. The UART was to be setup such that when any data was received on UART1 it had to be transmitted to UART2. This setup was carried out so that data transmitted through Bluetooth could be displayed on the LCD. Our main goal for this was to be able to display the ID3 data for the MP3. In order to carry out such an operation, all incoming data needed to be stored in a separate database. This was done to prevent the loss of data by overrunning the receive buffer of the chip. A FIFO was used to store all incoming data; this FIFO then became the source for the next transmission. The FIFO maintained the correct order of data – first in and first out. The main problem faced by us during this process was caused by the receive/transmit interrupt calls. The FIFO writes had to be interrupt protected so that data is not lost when a receive/transmit interrupt occurs during the write into the FIFO. This was done by incorporating the following Macro:

```c
unsigned int fifo_put(FIFO *f, const unsigned char c)
{
    char saved_ipl;
    ClrWdt();
    SET_AND_SAVE_CPU_IPL(saved_ipl,7);
    if (f->size >= fifo_capacity) // no room
        { RESTORE_CPU_IPL(saved_ipl);
          return false;
        }
    f->data[f->head] = c;
    f->head = (f->head + 1) % fifo_capacity;
    f->size++;
    RESTORE_CPU_IPL(saved_ipl);
    return true;
}
```

The Interrupt Priority Level (IPL) for that write sequence was first saved and then set to the highest level possible – 7. This ensured that the process could never be interrupted. After the write process was carried out, the original IPL was restored so that normal PIC operation could take place. Also to ensure that there wasn’t to much delay between the receiving and transmitting the information, as soon as the receive interrupt was completed we triggered a transmit interrupt for the other UART by setting its transmit interrupt flag to 1.

On a restart, the PIC was programmed:

1) Start the I2C protocol
2) Initialize the MP3 decoder chip
3) Close I2C
4) Start the UART for the Bluetooth receive and the LCD transmit
5) Start the SPI
6) Receive data from UART (transmitted through Bluetooth)
7) Transmit file through SPI to DOSon and store it in Flash card
8) Read file from Flash Card, once store is complete
9) Transmit to MP3 decoder via SPI.
LCD
In order to provide debugging output directly from the PIC, and also to enable features that may be added later on, we purchased and interfaced an LCD display. We took the main portion of the design from SparkFun.com (a hobbyist site which allows use of its designs for free for educational purposes). It was quite simple to use the LCD after the LCD controller was programmed and the UART port on the microcontroller was enabled. The code (which uses the CC5x compiler from www.bknd.com) and .hex file are both included in the copy of the SVN repository included with this paper. This .hex file can be used to program a 16F688 PIC (from Microchip) in a manner similar to the dsPIC described above. The schematic for the LCD controller follows:

DOSonChip
The mp3 player we designed was able to be updated via a Bluetooth link. The DOSonChip was part of the storage module that allowed us to store mp3 files on the mp3 player. We had a media card socket so that we could store the files on an SD card. The DOSonChip gave access to the card while providing a DOS-based environment. Both the DOSonChip and the Media Card Socket could not be bread-boarded directly; so we made a breakout board for them. Sparkfun provides a breakout board with the DOSonChip and the Media Card socket.
Schematic

Figure 5: DosOnChip Schematic

Board

Figure 6: DosOnChip Board
Communication buses

Bluetooth
Bluetooth is a short-range wireless communications protocol aimed at the networking of integrated systems. It is designed to be low-cost, low-power, and resistant to radio interference (through the use of band-hopping and fast acknowledgements). This protocol was the main focus of our project and was chosen to be the bridge between a PC or PDA to our MP3 device. On top of the Bluetooth protocol sit what are called “profiles” which are basically higher level protocols which use Bluetooth for data transfer. Our goal was to create a minimal implementation of the Advanced Audio Distribution Profile (A2DP) which provides for streaming of high-quality stereo MP3 over the Bluetooth connection. This later proved infeasible.

UART
The Universal Asynchronous Receiver/Transmitter bus is the de-facto standard for low cost serial data transfer. It uses two lines: one for transmit, one for receive. The baud rate on each end must be set to matching values for data to be transmitted. There are several options for this protocol, but the most common configuration is “8N1”. This indicates 8 bits of data followed by no parity bit and 1 stop bit. In this configuration, each value sent is preceded by a high voltage (5v) and followed by a low voltage (0v). A constant voltage represents no data at all.

RS232
RS232 is essentially the same protocol as UART with different voltage levels. Negative ten volts are used for logic ‘1’, and positive ten for logic ‘0’. In order to send data from a UART bus to an RS232 bus, a simple voltage converter is used. This chip is usually termed “MAX232,” but the part number of our converter is HIN232. Circuit diagram follows:
I2C
The Inter-Integrated Circuit (I2C) bus is designed to be a medium speed, low cost, short distance bus to connect various chips within a design. This bus was needed to send the configuration file of 2028 values to the MP3 decoder chip. On the bus is a single master and potentially several slaves. This is implemented with only two wires by using a “wired-and” scheme. Wired-and implies that the line is high if and only if all components connected to the line are outputting a high signal. To accomplish this, the two wires on the bus are connected via a pull-up resistor to logic 1. All the pins connected to the bus are in one of two states: high impedance, or logic 0. When all the connections to the bus are high impedance, the bus’ voltage is pulled up and a logic 1 is obtained. If any of the connections are logic 0, then the bus is pulled down. This scheme prevents any short circuiting if an error occurs and more than one chip tries to use the bus at once. One of the I2C lines is the data line (SDA) and the other is the clock line (SCL). In general, the master controls the clock line, but during a master read, the slave controls the clock line. After each 8 bits received, the receiver pulls the data line low. The sender then checks this 9th, and if the acknowledge is not present, the data is resent. The data line is not allowed to change when the clock is high, except under special conditions. If the data line makes a negative transition during a high clock, this is the “start” condition and indicates the beginning of a read or write operation on the bus. If the data line instead makes a negative transition, this is the “stop” condition, indicating the end of the operation.

SPI
The Serial Peripheral Interconnect bus is similar but much simpler than the I2C bus. This bus is a three-wire interface with only one slave and one master. There is a Master-Input-Slave-Output (MISO) line, a Master-Output-Slave-Input (MOSI) line, and a clock line.
At the heart of the SPI controller is a shift register. On each positive edge of the controller, the most significant bit of the register is shifted onto the output line, and the input line is shifted into the least significant bit. After each eight clock cycles, software must read out the data from this register (if the software wants SPI input) and a new value placed into the SPI register.

**Subversion**

Subversion (SVN) is a version control system replacement for the older Code Versioning System (CVS). A version control system aids developers in keeping each member up-to-date on changes in code, and aids in resolving any conflicts that arise, as well as many other useful functions.

**Simple Subversion work cycle**

1. Create a working-directory for your project
2. Check-out the project to this directory
3. Right click the newly created folder
4. Select “SVN Checkout”
5. URL = "svn://some.machine/folder"
6. Click OK
7. Update/Commit whenever the project is in a release-worthy state
8. Right-click the working-copy folder
9. Select “SVN Update"
10. You may need to resolve conflicts if the Trunk has changed
11. Select "SVN Commit" and enter a descriptive log message
12. Click OK
13. Repeat Step 3

**How to view the included repository**

Move/extract the copy of the repository on the CD to your desktop. After this, you can use the above instructions with the URL = “file:///W:/path/to/repository”.

**Further reference**

- SVN tutorial written by Buck Golemon⁹
- The official SVN reference¹⁰

**Recommended SVN downloads**

For Windows, TortoiseSVN¹¹ is by far the best SVN client. Definitely install it if you plan on using SVN. For other operating systems eSVN¹² has very similar features.

**Eagle**

Eagle is a free (as in beer) program to do PCB schematic and layout. You will want to keep all designs specific to your project in a single library, so the first step is to create an
eagle library. In this library you will create representations of any parts that will be on your PCB. Each part has three components: a symbol which will represent the part on the schematic, a package which will show up in the PCB layout, and a "Device" which is simply a mapping between the previous two. After all (or most of) the parts are created, a schematic is made which represents the circuit at the topological level. After the schematic is complete, Eagle can create a layout from this which has all the parts and connections of the schematic diagram. Your job at this point is to put all the parts in place, and to create all the traces (wires) and vias (holes) that will connect them. Once the layout is complete, a Design Rule Check (DRC) is performed to ensure that the manufacturing requirements are being met. Finally, "gerber" files are created from the layout to be sent off to the manufacturer. After this, all that's left is to wait to receive your PCB in the mail!

**Auto layout**

There is a nice tool in eagle to do automatic layout (Tools->Auto...), but there is some trickiness to making it work correctly. The DRC system (Tools->DRC...) holds all of the information about the capabilities of your manufacturer, and also can be modified to fulfill specific needs of your project. This information is used to do the automatic layout. If the automatic layout does not do what you want, then it is probably these settings that need to be modified. In particular, the in-house ECE PCB's have vias that are unplated, which means that you will need to solder microwire between the top and bottom layers to make the vias connected. This means that if your via is close to some other component, it will be quite difficult to not make an unwanted solder bridge. Also, if you have vias under a component, it will not sit flat to the board due to the microwire and solder, possibly making it impossible to solder the component down. There are several specific settings that you can set to avoid these pitfalls:

1. Under the "Clearance" heading in the DRC dialog, set the Wire-to-Wire and Wire-to-Pad clearance to 10mil since this is the minimum spacing for the ECE PCB's. Set the Pad-to-Wire clearance to 15mil to will make soldering components easier. Set the clearance for all 3 via settings to 30 mil to make the microwire soldering much easier.

2. If there are areas of your parts which should not have vias (such as under a quad flat package chip), you can create a via keep-out area. Open the package of the part and select the "Rect" tool. Set the layer to "vRestrict" and any area that you draw will not have vias.

**PCB**

**In-house PCB**

The In-house PCBs are made very fast - you'll generally get them within 2-3 days. The PCBs produced can only have two layers. Also, they use a special process due to which the boards dont have a solder mask. The lack of a solder mask makes it easier to create bridges. Additionally, the vias are not plated. This means you'll have to fit a wire through the via and solder it to the top and bottom layers of the PCB. Because of this, make sure that you don't place any vias under chips.
External PCB Manufacturers
If you need to make more complicated PCBs, we found a couple of good places -
- BatchPCB (www.batchpcb.com) offers very cheap PCBs - about $2.50 per square inch. The main advantage of this place over the in-house ones is the presence of a solder mask and plated vias. The major disadvantage is the lead time.
- $33 PCB (4pcb.com) is more expensive but has smaller line/space and hole size specifications. Additionally, although the site says minimum quantity is 3, students can order a single PCB.

General Advice

Documentation
Ensure that you always have the documentation (datasheet, specs, etc.) for your parts. If you are working with industry standards and protocols, it becomes almost impossible to accomplish anything without those documents.

Utilize widely-used protocols
A2DP was a very new protocol and hence there was little information available. Additionally, not many sites provided code on how to implement A2DP - even the Linux BlueZ Bluetooth stack was not very helpful. In a year, A2DP will become more established and the resources available for the protocol would be much larger.

Vias on in-house PCBs
The in-house PCBs don't have plated vias. This means that there's no connection between the top and bottom layers through a via. To make the connection, we had to solder a wire on the top and bottom halves. Make sure you don't have any vias under your chips. Additionally, try to minimize the number of vias you use.

Additional Chips
Always have additional chips on hand in case the one you're using burns out. This is especially true when its close to the open house. Also, try to use chips from the parts shop as much as possible - since it will be easier to get additional ones from there. If you need to order parts, do it early and order more than you think you'll need. Microchip offers free samples, try getting your PIC chips that way. Also, it is not always a good idea to use parts from previous projects. It might be hard for you to test its functionality and you might end up with something previously damaged.
References

1 Bluetooth SIG, GAVDP v1.0, fig. 2-1, p 11
2 Bluetooth SIG, Core v2.0, System Architecture, fig 2.1, p 95
3 Bluetooth SIG, Core v2.0, HCI specs, fig 1.1, p509
4 http://www.bluetooth.com/Templates/MT5_TechnicalSpecificationListing.aspx?NRMODE=Published&NRORIGINALURL=%2FBluetooth%2FLearn%2FTechnology%2FSpecifications%2F&NNODEGUID=%7B8637C7FE-3DD6-4437-A30D-798E014DC055%7D&NRCACHEHINT=NoModifyGuest
5 Present in accompanied CD
6 http://www.sparkfun.com/datasheets/BreakoutBoards/DOSonCHIP-SD-Schematic.pdf
8 http://www.sparkfun.com/datasheets/Prototyping/SD-Socket-PP-14446.pdf
9 http://legend.me.uiuc.edu/wiki/index.php/Subversion
10 http://svnbook.red-bean.com/
11 http://tortoisesvn.tigris.org/
12 http://esvn.umputun.com/