Bandwidth Unlimited
School of Engineering Science
Burnaby, BC
V5A 1S6
bandwidth-unlimited@sfu.ca

November 5, 2001

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, BC
V5A 1S6

Re: ENSC 340 Design Specifications for MRx Home Theatre Interface

Dear Dr. Rawicz:

The attached document, *MRx Home Theatre Interface Design Specifications*, outlines the design specifications of our project for the Engineering Science Project Course, ENSC 340. Our project is a system that allows people to play MP3 sound files on their home theatre systems.

The design specifications describe how we will implement the requirements discussed in our functional specifications. Within the document, our three main components (the host computer, the MRx and the Palm Pilot) are discussed in detail. Also discussed are our hardware and protocol design.

Bandwidth Unlimited is a team comprised of five highly skilled and talented engineering science students: Mavis Chan, Brian Fraser, Manpreet Gakhal, Ben Lake, and Gabrielle Sheung. If you have any questions or concerns about our proposal, please do not hesitate to email us at bandwidth-unlimited@sfu.ca, or to phone me at (604) 298-6442. Thank you for your time.

Sincerely,

Gabrielle Sheung
Bandwidth Unlimited

Enclosure: MRx Home Theatre Interface Design Specifications

Bandwidth Unlimited

MRx Home Theatre Interface Design Specifications

Project team:

Mavis Chan
Brian Fraser
Manpreet Gakhal
Ben Lake
Gabrielle Sheung

Contact Personnel:

Ben Lake – benl@sfu.ca

Submitted To:

Dr. Andrew Rawicz – ENSC 340
Steve Whitmore – ENSC 305
School of Engineering Science
Simon Fraser University

Issue Date:

November 5, 2001

Revision: *1.0*

Executive Summary

The MRx Home Theatre Interface is an innovative project, testing the mettle of this Ensc340 group. The scope of the project is large so as to achieve the sophisticated task of streaming high quality MP3 encoded music to any home theatre system for all to enjoy.

Much thought and consideration was put into the planning of the project to ensure that the MRx Home Theatre Interface is a useful, inventive and exciting new product to bring to the electronics market. Several products are already commercially available that rival the MRx. However, we know we can produce a far superior product to those existing in the market already. With several key additions, we are confident MRx will gain a profitable niche in the MP3 audio marketplace.

This document outlines methods of implementing the requirements Bandwidth Unlimited has decided upon in the functional specifications. These implementation discussions include both high and low levels of the designs and are categorized in the same manner as the requirements in the functional specifications document. Detailed are the protocols to be used, design of the user interfaces at each of the devices (host computer, MRx and Palm Pilot), file server, MPEG decoding and our choice of hardware.

Table of Contents

Acronyms & Terms

API –	Application Program interface
CD-ROM –	Compact Disk Read Only Memory.
EDO –	Extended Data Output
EEPROM –	Electrically Erasable Programmable Read Only Memory
Glueless –	No additional circuits required to interface parts together
IrDA –	Infrared Data Association.
LCD –	Liquid Crystal Display
MHz –	Megahertz.
MP3 –	MPEG (Moving Picture Experts Group) Audio Layer 3.
MRx –	MP3 Receiver. (The device being designed and built by Bandwidth-Unlimited.)
OS –	Operating System.
OSS –	Open Sound System
PC –	Personal Computer.
PCM –	Pulse-Code Modulation
RAM –	Random Access Memory.

1. Introduction

In the past few years, MP3 compression technology has refashioned the way we enjoy our favourite music. New file sharing programs allowed people to create song libraries of enormous proportion on their computers. With such a vast array of high-quality digital music, it is perplexing that people have only been able to enjoy this music on a pair of mediocre computer speakers.

The MRx Home Theatre Interface is a system that will enable people to listen to digital music on their home theatre system. With the MP3 player interface to the home theatre system, people can finally play their favourite songs with the audio quality they deserve in any room of their home.

This document will present the implementations we decided upon according to the requirements we feel are necessary to allow users this freedom. This document is divided into the major portions of our project (the host computer, MRx and the Palm Pilot) as well as the protocols we will be using to program the devices. The user interfaces are described in detail in the host computer and Palm Pilot sections while the MRx section also includes our choice of hardware and other implementations.

2. System Overview

The MRx system will allow people to play MP3 sound files stored on the host computer on their own home theatre systems. Figure 2.1 below illustrates the basic structure of this system. The host computer contains the library of MP3 files, and will send a stream of MP3 encoded audio to the MRx device via an Ethernet connection. The MRx device will deliver the decoded sound to an adjacent home theatre.

A two-way infrared link between the Palm Pilot device and the MRx allows the Palm Pilot to be used as a remote control for the MRx. The user interfaces on the host computer and on the Palm Pilot device allow the user full control over the music selection and playback, and provide the user with updated status information from the MRx. The front panel of the MRx device also allows for limited control over the music playback and provides the user with graphical feedback via an LCD screen.

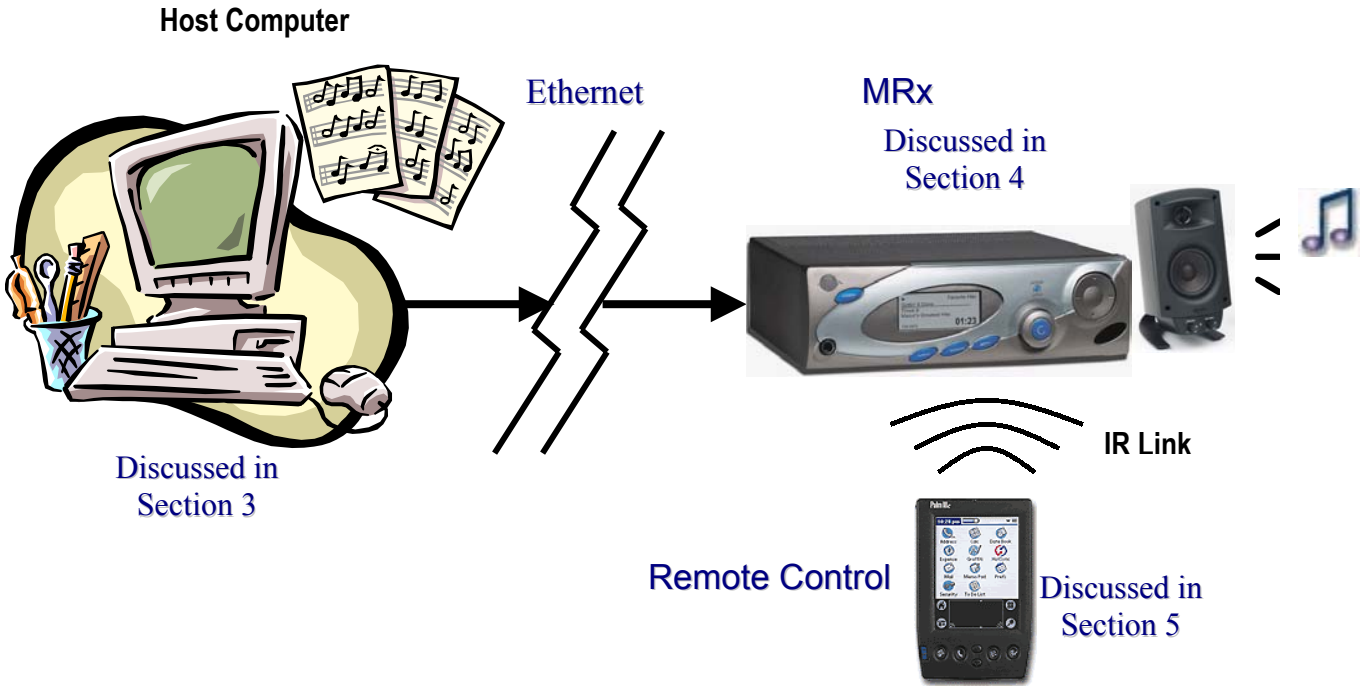

Figure 2.1: System Overview

3. Host Computer

The following sections detail the designs specifications for the host computer portion of our project.

3.1 PC User Interface

3.1.1 High Level Design

The graphical user interface of the PC Host provides basic audio and playlist controls. The basic audio control operations, such as play, stop, pause, resume, skip forward, skip backward, return to start of song, and go to next song, are controlled by the clicking buttons on the application. The user interface gives feedback to the user by displaying the current MP3 track information and the current audio control operation performed on the track. Additionally, the user interface indicates where the current position of the MP3 track is in terms of its timing information. The playlist controls are similarly controlled by clicking buttons on the application and allowing the user to add, edit, or remove playlists. The user interface provides feedback to ensure that the user does not accidentally perform conflicting operations.

Physically, the first layer of the GUI will be about twice the length vertically as it is horizontally. This configuration allows the MRx application to be moved to the sides of the PC monitor without obscuring the majority of a 800X600 (min.) resolution screen. The top half of the application will be used to provide the basic audio controls, while the bottom half of the screen will be used to provide the playlist controls. The second layer of the GUI will be several different smaller user interface window which allows the user to multi-step operations such as adding MP3's, load playlist, or remove playlists.

The features of the audio control portion of the application are listed in Table 3.1:

Table 3.1 – Features of audio control part of PC GUI application

Feature	Description
Status indicator	Graphically illustrates the state of the MRx - play, stop, pause, forward, or backward.
Track info window	Displays the track number, name, artist, length, and sample rate inside a text box.
Backwards button	When pressed once, the MRx will return to the beginning of the track. When pressed continuously, the MRx will move backwards to a previous location of the track.
Play button	Starts playing the current track.
Pause button	Stops playing the current track. When play button is pressed or pause button is pressed again, the track will resume playing from the previous location.

Stop button	Stops playing the current track without remembering where the track was stopped.
Forward button	When pressed once, the MRx will skip to the next track. When pressed continuously, the MRx will move forwards to a future location of the track.
Random button	Allows the MRx to play the tracks in the playlist in a random order.
Time position indicator	Graphically illustrates the current time location in the MP3 track in relation to its entire track length.

The features of the playlist control portion of the application are listed in Table 3.2:

Table 3.2 – Features of playlist control part of PC GUI application

Feature	Description
Playlist window	Displays the track information for all tracks in the current playlist. This list will be dynamic such that the user can select a particular track and move or remove the track, or utilize the audio control buttons to start playing the selected track.
Add button	Opens a separate application window that displays the subdirectories and available MP3 files in the current directory. The user can select the files to add to the playlist, or select a directory to traverse the directory tree.
Remove button	When a track in the playlist is selected, the track is removed from the current playlist.
Save button	The currently displayed playlist is saved, or the user is prompted for a playlist name if the playlist is new.
Load List button	Opens a separate application window that displays the subdirectories and available playlist files in the current directory. The user can select the playlist to load to the main application, or select a directory to traverse the directory tree.
Remove List button	Opens a separate application window that displays the subdirectories and available playlist files in the current directory. The user can select the playlist to remove, or select a directory to traverse the directory tree.
Create New button	Clears off the playlist window and creates a blank playlist for the user to add MP3 files as desired.

3.1.2 Low Level Design

The GUI application is responsible for taking the information the user requests from the GUI and creating a packet to be sent to the MRx module and handled. This packet will generally contain a request for data or a playlist file (in the case when the playlist is

saved). The GUI application will then expect a packet returned containing data that the application will unpack and display to the user on the GUI.

To make sure that the MRx has gotten the messages that the GUI sends, the GUI will wait for an ACK (acknowledge) from the MRx. If the MRx hasn't sent the ACK within a certain amount of time, the GUI will resend the message. After a predefined number of retries, the GUI will stop and notify the user that the MRx is not responding.

The playlist contents seen by the user on the GUI is always saved to a file containing the current playlist. Anytime the playlist is changed, the playlist is saved to Current_Playlist.pls. If the user only wants to play a single MP3 file, the current playlist will contain that single MP3 file. The only time the actual playlist file is saved is when the user explicitly requests the playlist to be saved. Then, the current playlist is copied to the appropriately named file.

3.1.2.1 Audio Control

To implement the audio controls functionality such as play, stop, resume, skip forward, skip backward, return to start, and go to next, the GUI will send a packet requesting the particular action. The GUI application will then expect a status packet returned from the MRx such that it can then proceed to update the status indicator of the GUI.

3.1.2.2 Get List of Playlists or List of MP3 files

To get a list of playlists or a list of MP3 files, the GUI will send a packet requesting a list of a particular filetype and a list of the subdirectories of the current directory. The current directory is either default or selected by the user from a list. The GUI application will expect a packet returned containing the list of playlists or files, and the list of subdirectories. The GUI will then proceed to display the available subdirectories, and playlists or files.

3.1.2.3 Get Playlist

To get a playlist, the GUI will send a packet requesting a particular playlist by name and path. The GUI application will expect a packet returned containing the playlist. The GUI will then proceed to display the playlist in the playlist window.

3.1.2.4 Add MP3 File

To add a MP3 file, the GUI will send a packet requesting a particular MP3 file by name and path. The GUI application will expect a packet returned containing the MP3 file information. The GUI will then proceed to add the MP3 file information to the current playlist.

3.1.2.5 Update Playlist (Save, remove MP3 or change order of current playlist)

To update the playlist, the GUI will send a request to send the updated playlist. A packet will be constructed containing the new playlist. The GUI application will expect a packet returned indicating the update is done or else requesting confirmation to overwrite the playlist. If a confirmation is needed, the update will be tried again with the user's response. If a save is explicitly requested and the playlist is being saved for the first time, the user will be asked to provide a file name.

3.1.2.6 Delete Playlist

To delete a playlist, the GUI will send a packet requesting a particular playlist by name and path to be deleted. The GUI application will expect a packet returned indicating the removal of the playlist is done. Since the playlists are copied to Current_Playlist.pls and this file can never be deleted, there will be no conflict with deleting a playlist that is playing.

3.1.2.7 Create New Playlist

To create a new playlist, the GUI will clear the playlist window, which essentially clears the current playlist file. The user will add MP3 files as detailed in the procedure above. Then the user will request an update playlist (save) to save the current playlist to an actual playlist file.

3.2 File Server

The host PC will run a file server application, which will be responsible for maintaining and transmitting all of the relevant playlist and MP3 information. An Ethernet connection with the MRx device will allow the file server to communicate and transfer data with the MRx control software. Lists of playlists, lists of MP3 songs, playlist information, and MP3 data will be transferred between the file server and the MRx, with only a single file transfer occurring at any one time.

The file server will store all information using a virtual directory structure that will be configured by the user. The root directory will be specified by the user, and for privacy and security reasons, access to directories above this root directory will not be allowed by the file server. All playlists will be stored in the '/Playlists' directory which will not allow for subdirectories. MP3 files may be stored in any subdirectories created within the root directory, and the user may browse directories within the root directory.

The file server application will display information regarding files currently being downloaded, and logs of files that have previously been downloaded. This information will be used primarily for debug purposes during implementation of the design.

We considered three different options of implementing the data transfer protocol between the file server and the MRx: integrating an existing FTP program with our software,

using an existing library of FTP C functions from within our software, or writing our own FTP protocol. With the first option, we would not have to write as much code ourselves, but we would have less control and it may become difficult to integrate and use the existing software with our own software. Similarly with the second option, we would not have to write as much code ourselves, but it could be difficult to completely understand how the already defined functions work and use them with our own software. The third option, writing our own protocol, would require more time to write the code ourselves, but we would have complete control over the implementation and it would be easier to integrate with our other software. We decided that the option of writing our own protocol would be the easiest to implement in our file server design.

4. MRx

The following sections discuss the design specifications for the MRx, our main device.

4.1 MRx Hardware Design

The hardware for the MRx device contains all of the necessary features in order to operate as a fully independent MP3 Player that is capable of two-way communication with a host computer and a graphical remote control.

This section shall detail the hardware architecture and design features required to implement the MRx hardware. The block diagram for the MRx hardware architecture is shown below in Figure 4.1.

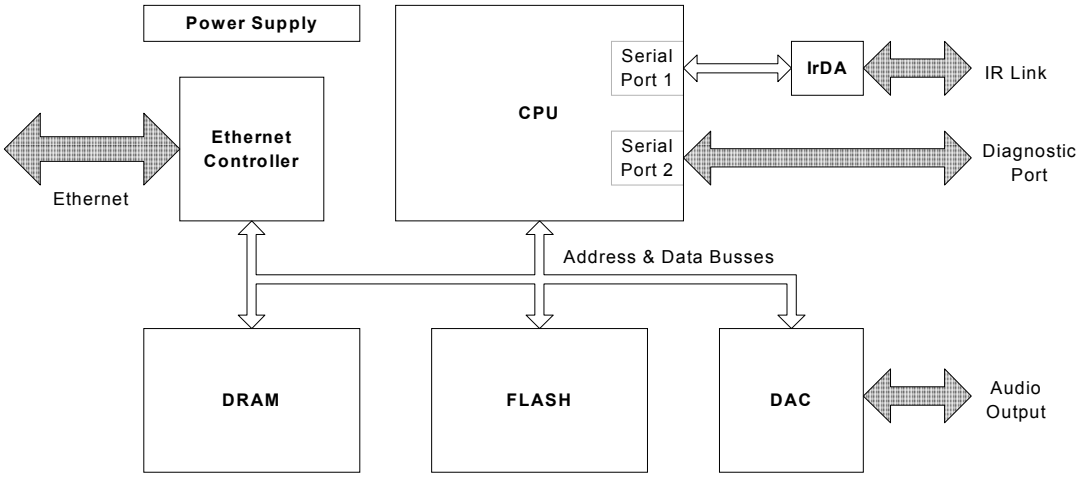

Figure 4.1 – MRx Hardware Architecture Block Diagram

4.1.1 Central Processing Unit (CPU)

The CPU for the MRx was required to be able to decode MP3 files in real time using software algorithms, as defined in the *MRx Functional Specifications*. The Cirrus Logic EP7212 processor was selected since it satisfies that requirement. Other processor families that were evaluated included the Motorola 68000 Series, which allow for similar capabilities in processing power. However, most development environments for the Motorola family processors are proprietary and require purchased licenses for use. The ARM family of processors also offer comparable processing power, and use Linux, an open source operating system, which offers free development environments.

The Cirrus Logic family was selected since they are based on an ARM core, yet offer integrated sub-systems that were similar to our project requirements. The processor datasheet specifies that the CPU will use 50-70% of its runtime to decode an mp3 file in

real-time, allowing an acceptable amount of overhead for other data processing and message management.

4.1.2 Ethernet Interface

The Ethernet interface of the MRx will be comprised of a Cirrus Logic CS8900A Ethernet Controller. The controller provides all required functionality for a 10 Base T Ethernet interface and requires no additional logic to interface to the processor busses. No serial PROM is required to configure the controller, since it is not a requirement that the Ethernet connection be operational on power up. The CPU shall configure the Ethernet controller during the initialization phase of board start up.

Other interface standards such as USB and high-speed serial were investigated during the initial planning phase of the project. However, no other interface is comparable to Ethernet for the combination of speed, multi-drop capability, and industry wide acceptance that Ethernet offers, for the MRx application

4.1.3 IR Interface

The design of the IR interface was chosen to use the IrDA standard for IR communication. There is no other comparable standard for IR data communication, and developing a new standard was estimated to be too much work. The IrDA standard specifies a maximum distance of 1m. The hardware development for this product will include research and testing to extend the communication distance to at least 3m.

The EP7212 features a fully integrated IrDA coder/decoder. The only remaining component of the IR interface is the physical LED and photodiode, which is available in a single package. The TFDS4500 from Vishay Electronics will be the physical IR transceiver on the board, with additional non-populated spots on the board reserved for parallel transceivers, in the event that a single transceiver does not provide the required range as defined in the *MRx Functional Specifications.*

The IR interface shares serial port 1, and a standard DB9 connector will also be installed on the board in order to allow serial program download during the programming mode. Appropriate software will be used to select the desired input source to serial port 1, depending upon the mode of operation.

4.1.4 Memory

4.1.4.1 Boot Code

The MRx motherboard does not require boot code to be externally supplied to the CPU. A jumper will be located on the board to place the CPU into boot mode, in order to initially program the board. The CPU has hard-coded internal instructions which it executes during boot mode that force it to wait for program download from serial port 1 when starting up.

4.1.4.2 Non-Volatile Memory

The application code for the MRx needs to be retained in non-volatile memory, such that it can correctly restart after being powered down. The application program is not expected to exceed 8MB in size. These requirements of high-density, non-volatile storage suggest that FLASH memory would be feasible. There are few other options to consider when selecting the non-volatile storage elements, since EEPROM has a very slow programming time and low density, while hard drives require a relatively complex interface to the main data bus.

The MRx shall be equipped with two Atmel AT49LV320 2Mx16 bit FLASH memories to satisfy the non-volatile memory needs of the board.

4.1.4.3 Volatile Memory

During program operation, it has been estimated that 16MB of volatile memory will be required to store real-time mp3 data and program variables. The read/write accesses to volatile memory will be much more frequent than to non-volatile memory, so a faster memory technology is required. Dynamic RAM (DRAM) satisfies the volatile memory requirement of the board since it is a very dense, very fast, and reasonably affordable memory technology.

There were no other options considered for the volatile memory because the Cirrus Logic processor contains an on-board DRAM controller. Therefore, it made logical sense to use the memory type best suited to the selected processor. The MRx shall be equipped with two Hitachi HM5165165 4Mx16bit EDO DRAM memories to satisfy the volatile memory requirements of the board.

4.1.5 Power Supply

To reduce design complexity, an external power adapter was chosen to power the board, as opposed to building an on-board 120VAC power supply. An AC wall adapter supplying 6-12VDC up to a maximum of 1A shall power the MRx. Onboard linear and switching regulators shall supply the 5V reference voltage for some components, the 3.3V I/O voltage, and the 2.5V core voltage for the. All available design tricks will be used to ensure that the power supply is designed to be as stable and as noise-free as possible.

4.1.6 Digital to Analog Converter (DAC)

The DAC shall receive digital information from the CPU and convert it to an analog signal that will be supplied to RCA style jacks for output. To satisfy the audio requirements outlined in the *MRx Functional Specifications*, the selected DAC is the Cirrus Logic CS4340. Several other DACs with similar specifications could have been located, however the CS4340 offered a glueless interface to the main data bus, and was

readily available. The selected DAC operates stereo output channels, with a maximum signal rate of 96kHz and a resolution of 24-bits.

For line-level audio output, which is the specification for the MRx board, no external audio amplification is required after the DAC.

4.1.7 Liquid Crystal Display (LCD)

The MRx requires some form of output to display information such as current status, music information, and menu data. Two different types of LCDs were evaluated for the MRx device. A graphical LCD gives complete pixel-by-pixel graphics in modes such as 180x120, 320x180, and higher. A textual LCD has multi-line character display abilities for pre-programmed characters.

Considering the possibility that the MRx will reach the consumer market one day, the character-based LCD was chosen since it costs ~$50, while graphical LCD displays are ~$150. A graphical LCD would cause the retail price of the MRx to be excessively high compared to competing products, for very little gain in functionality.

4.1.8 Printed Circuit Board (PCB)

The PCB required to hold the high-speed and high-density components of the motherboard has considerable requirements. The pitch (pin-to-pin spacing) of at least 6 components on the board is 0.5mm, with pin counts for these components being anywhere between 48 and 208 pins. The fine pitch of these components make it almost impossible to hand assemble a PCB. Bandwidth Unlimited has a group member capable of hand assembling 0.5mm pitch components, however the laboratory does not contain the soldering and rework equipment to properly assemble a board of this complexity.

The busses in the MRx architecture are specified to operate at approximately 74MHz. To accommodate busses as wide as 32-bits at that speed, it is estimated that 4 to 6 signal layers are required, as well as two more layers for power and ground. The additional power and ground planes are essential in high-speed high-density architectures to provide shielding and prevent cross-talk between traces. The cost of one PCB panel, based on a 6 or 8 layer design was quoted to be over $1200CDN.

Due to the excessive cost and high-density required on the MRx motherboard, the development of the PCB will have to be delayed until a later stage of the project, when additional funding can be raised. The prototype demonstration, occurring in December 2001, will be conducted from the EP7212 development board that is on loan to Bandwidth Unlimited. Several hardware additions, such as a character based LCD screen and an extended range IR module will need to be attached in order to satisfy the prototype design requirements.

4.2 MRx Software Design

The MRx runs BlueCat Linux 3.1. Under Linux, the MRx software is built as a multithreaded application. The software is object oriented based, and most control and data structures are inside C++ classes. All of the software runs in user mode (non-kernel mode). This means that there are no device drivers needed.

4.2.1 Thread Definition

Threads are very similar to a standard executable processes because they runs in "parallel" just like multiple processes do. However, the difference between having a multithreaded program and a multi-process program is that all threads of a program share the same memory area. Therefore, with threads, one can pass pointers from one thread to another; with processes one cannot. Threads are central to the flow of the program, and the shared memory space is exploited for downloading, storing, decoding and playing MP3 files.

4.2.2 Message Passing Between Threads

It was found that Linux does not have a suitable method to pass messages between threads, therefore our own method must be created. A class named CMessage is made such that when a message needs to be sent, an instance of this class is created and sent (see Sending a Message below). Also, the code is able to spawn threads, so it cleans up starting and stopping threads (see Thread Processes below).

4.2.3 Threads Used

The application uses use 7 threads:

Thread Name	**Description**
Menu	First thread to be launched. Spawns all sub threads and displays an interactive text menu.
Control	This is the main thread that does all the work between different threads. It will process almost every message sent or received by the MRx.
MP3 Decoding	Decode and play the MP3 file
MRx Interface	Receive physical button presses on the MRx interface
PC Play Control	Receive play control commands from the PC
IR Control	Receive play control commands from the Palm
Fileserver	Receive files from the File Server

Each of these threads is shown below in Figure 4.1. This figure shows which threads communicate with which other threads, and the general design structure of the MRx code.

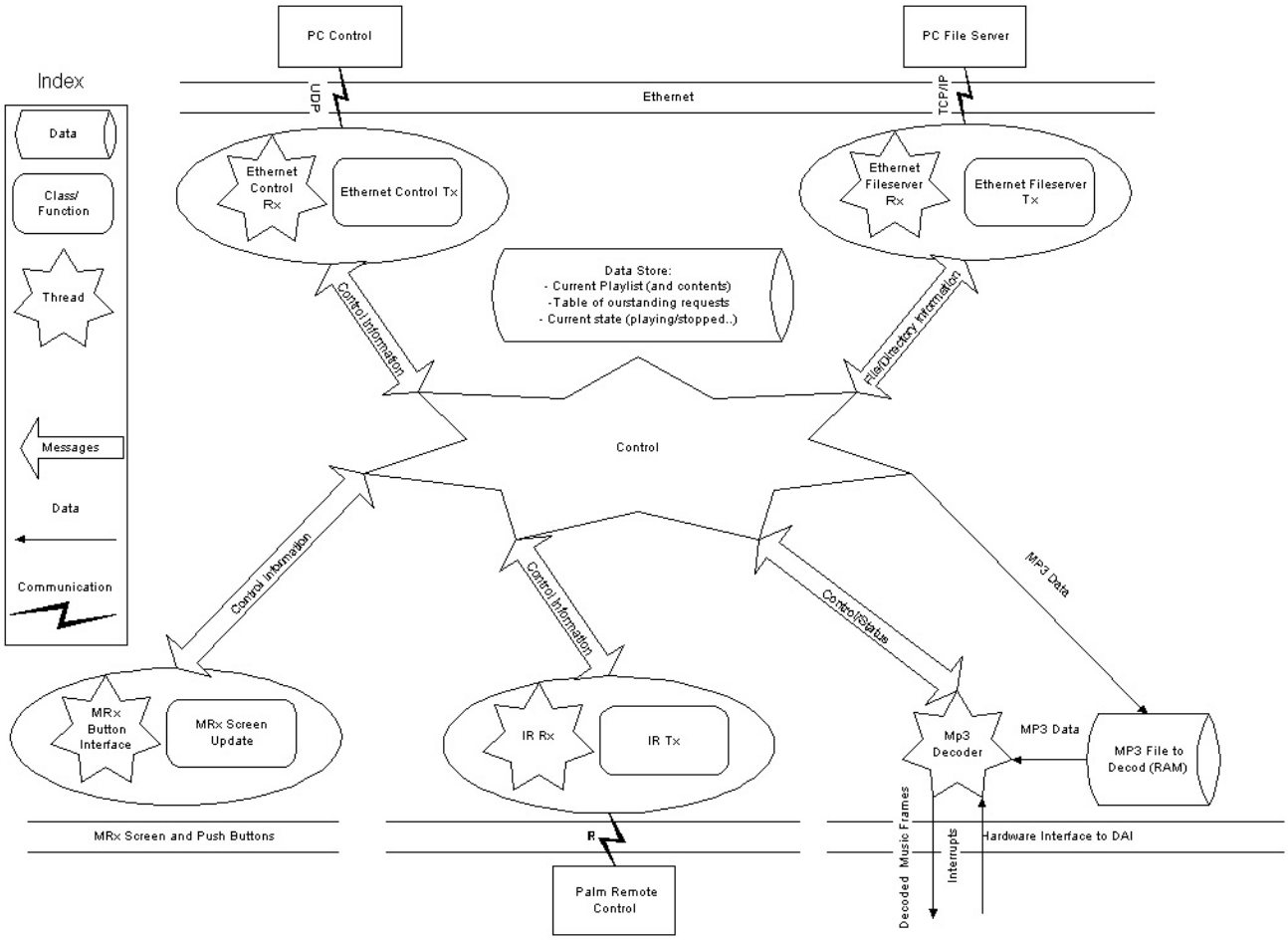

Figure 4.1 - MRx Software Threads

4.2.4 Class Design

The basic structure of the class architecture (from the top down) is:

CThreadMaster

- This class controls spawning all the threads, and coordinating all inter thread communication.
- It also provides a central access point to all the threads. For example, to check what threads are running.

CThread

- There is one of these classes for each thread.
- It tracks the current state of the thread.
- It tracks which thread it is (IR receive thread, Fileserver receive thread).
- It has a message queue for messages waiting to be processed by the thread

- It has a the code needed to implement the message passing. This is the funny use of the mutexes.
- Since multiple threads may call this class at once (send a message and receive a message), it is thread safe (i.e., uses mutexes).

CMessage

- This is the lowest level of class.
- It stores the data to be passed back and forth between threads
- It may support some calls to change a message into a packet to be transmitted.

4.2.5 Global Operation

In global.h, there is a pointer to the one and only CThreadMaster class called g_pThreadMaster. This object is initialized by the main thread. After initializing this, the main thread spawns all the sub threads, and then enters the interactive text menu.

4.2.6 Thread Processes

The main thread (the text menu thread) runs on the file mrx.cpp. First the thread initializes the one and only CThreadMaster object called g_pThreadMaster in global.h. Then it calls SpawnSubThreads() to spawn all the sub threads. These threads will start running on the thread procedure (first function to be called by the thread). These procedures are all in the files starting with "thread_... .cpp".

Each thread procedure is passed a pointer to a CThread object called pMyThread. This is the CThread object for that thread. When the thread wants to receive a message, it calls pMyThread->ReceiveMessage() as follows:

```
CMessage* pMessage = NULL;
pMessage = pMyThread->ReceiveMessage();
// <... process the message...>
delete pMessage;        // Free the memory used by the message.
pMessage = NULL;
```

When the thread finishes, it calls pMyThread->SetFinished().

4.2.7 Sending a Message

To send a message from on thread to another (or even to the same thread), allocate a new message with "new", setup the data in the message and call SendMessage.

This is done as follows:

```
CMessage* pMyMessage = new CMessage;
pMyMessage->SetWhoFrom(...);
pMyMessage->SetWhoTo(...);
pMyMessage->......
SendMessage(pMyMessage);
```

```
// Note: do not delete the memory allocated here!
// It is deleted later when the message is received by the
receiving thread.
```

4.3 MPEG Decoding

The following sections discuss how the MP3 files will be received and decoded by the MRx for playing.

4.3.1 High Level Design

The MRx must be able to stream music to the home theatre system for playing. In order to do so, the MP3 data must be sent to a specific location for the MRx decoder to retrieve from and process. After decoding the MP3 data into a raw data format, the information is then passed to a driver to be play

4.3.2 Low level Design

MP3 files are coded into frames, of which there are about 38 in each second. At the beginning of each frame, 32 bits are reserved for the header frames. In the header frames, information about the data contained in the frame is stored. Table 4.1 lists these bits and their purposes.

Table 4.1 – Information contained in the header frames

Position	Purpose	Length (in Bits)
A	Frame sync	11
B	MPEG audio version (MPEG-1,2, etc.)	2
C	MPEG layer (Layer I, II, III, etc.)	2
D	Protection (if on, then checksum follows header	1
E	Bitrate index (lookup table used to specify bitrate for this MPEG version and layer)	4
F	Sampling rate frequency (44.1kHz, etc., determined by lookup table)	2
G	Padding bit (on/off, compensates for unfilled frames)	1
H	Private bit (on/off, allows for application-specific triggers)	1
I	Channel mode (stereo, joint stereo, dual channel, single channel)	2
J	Mode extension (used only with joint stereo, to conjoin channel data)	2
K	Copyright (on/off)	1
L	Original (on/off)	1
M	Emphasis (respects emphasis bit in the original recording; now largely obsolete)	2

The MRx will make use of this information to decode the given data and pass it on to a Linux driver called the OSS (Open Sound System). OSS is not capable of playing MP3 files directly, but can play PCM (Pulse-code modulation) files, which is why a decoder is necessary to process the MP3 file before it can be played.

OSS is already installed in our operating system and so is its API (application program interface). The API is needed to control the hardware through our program, to play songs through the audio output. For the OSS, the API is a file named `soundcard.h`. With this header file, various parameters can be set, such as sample rate, fragment size and stereo versus mono play mode. For example, the following segment of code opens up the audio output port and sets the mode to stereo and sample rate to 44.1kHz:

```
if ( (handle = open("/dev/dsp",O_WRONLY)) == -1 ) {
        perror("open /dev/dsp");
        return -1;
    }
if ( ioctl(handle, SNDCTL_DSP_STEREO, 1) == -1 ) {
        perror("ioctl stereo");
        return errno;
    }
if ( ioctl(handle, SNDCTL_DSP_SPEED, 22050) == -1 ) {
        perror("ioctl sample rate");
        return errno;
    }
```

Note: WRONLY stands for write-only. Since the port is bi-directional and the MRx only needs to write to it, this option must be set to prevent errors and noise. Also because stereo mode is chosen, the sample rate is 22050Hz instead of the 44100Hz because sample rate is calculated per channel and in stereo mode, the rate is divided evenly between the two channels.

After preparing the port, decoded data can be written to the port by the `write` command, with the data to be written and port destination as parameters.

4.4 MRx Interface

The MRx design will allow the user to perform a limited number of play control functions via push buttons located on the device. As stated in section 4 of the *MRx Functional Specifications* document, the MRx unit will contain at least six push buttons that will control the following basic operations: play, stop, pause, skip forward, skip backward, and fast forward. The MRx control software will be interrupted when a user button is pressed and the software will perform the corresponding control operation on the currently selected song.

The MRx interface will also provide visual feedback to the user through a LCD screen. Because large, high resolution LCD screens can be expensive, our interface design will support a minimum LCD size of two 40 character wide lines. Textual information such as the current status of the device and the name of the currently selected song will scroll across the LCD screen. In addition, since the user buttons on the initial prototype may not be labelled, the LCD will be used to tell the user what control operation each button corresponds to.

In the initial phase of development, the MRx front panel interface will provide only these few basic control operations and simple textual feedback through an LCD, but future improvements of the system could include a more comprehensive interface on the MRx device.

5. Palm Pilot

The following sections details the design specifications for the Palm Pilot portion of our project.

5.1 Palm User Interface

5.1.1 High Level Design

The graphical user interface of the Palm provides basic audio and playlist controls. The basic audio control operations, such as play, stop, pause, resume, skip forward, skip backward, return to start of song, and go to next song, are controlled by touching buttons on the application. The user interface gives feedback to the user by displaying the current MP3 track information and the current audio control operation performed on the track. Additionally, the user interface indicates where the current position of the MP3 track is in terms of its timing information. The playlist controls are also controlled by the touching of a button on the application and allow the user to add, edit, or remove playlists. The user interface also provides feedback to ensure that the user does not accidentally overwrite playlists.

Physically, the GUI will fill the entire Palm screen. The first layer of the application will be used to provide the basic audio controls, while the second layer of the application will be used to provide the playlist display and activate playlist controls. The actual playlist controls are implemented as a third layer.

The features of first layer of the application are listed in Table 5.1:

Table 5.1 – Features of first layer of Palm GUI application

Feature	Description
Status indicator	Graphically illustrates the state of the MRx - play, stop, pause, forward, or backward.
Track info	Displays the track number, name, artist, length, and sample rate inside a text box.
Backwards button	When pressed once, the MRx will return to the beginning of the track. When pressed continuously, the MRx will move backwards to a previous location of the track.
Play button	Starts playing the current track.
Pause button	Stops playing the current track. When play button is pressed or pause button is pressed again, the track will resume playing from the previous location.
Stop button	Stops playing the current track without remembering where the track was stopped.

Forward button	When pressed once, the MRx will skip to the next track. When pressed continuously, the MRx will move forwards to a future location of the track.
Random button	Allows the MRx to play the tracks in the playlist in a random order.
Time position indicator	Graphically illustrates the current time location in the MP3 track in relation to its entire track length.
Playlist button	Opens the second layer of the Palm GUI to view and manage playlists

The features of the second layer of the application are listed in Table 5.2:

Table 5.2 – Features of second layer of Palm GUI application

Feature	Description
Playlist display	Displays the playlist name and the track information for all tracks in the playlist. This list will be dynamic such that the user can select a particular track and move or remove the track, or click to start playing the track
Add button	Opens a separate application layer that displays the subdirectories and available MP3 files in the current directory. The user can select the files to add to the playlist, or select a directory to traverse the directory tree.
Remove button	When a track in the playlist is selected, the track is removed from the current playlist.
Save button	The currently displayed playlist is saved, or the user is prompted for a playlist name if the playlist is new.
Load List button	Opens a separate application layer that displays the subdirectories and available playlist files in the current directory. The user can select the playlist to load to the main application, or select a directory to traverse the directory tree.
Remove List button	Opens a separate application layer that displays the subdirectories and available playlist files in the current directory. The user can select the playlist to remove, or select a directory to traverse the directory tree.
Create button	Clears off the playlist window and creates a blank playlist for the user to add MP3 files as desired.

5.1.2 Low Level Design

Since the functionality of the PC user interface and the Palm user interface was intentionally designed to be identical for simplicity, please refer to section *3.1.2 Low Level Design* for the details of the Palm GUI low level design.

6. Protocol Diagrams and Specification

The following sections details the protocol used in this project.

6.1 Packet Specification:

This section defines the byte format of any message transmitted by across the Ethernet and IR interface. It covers communication between the MRx, PC Control Application, PC File Server and Palm Pilot.

6.1.1 Packet Layout:

Packet Type	(4 characters)
Command	(4 characters)
ID Number	(4 characters)
Fragment Number	(4 characters)
Data Length	(4 characters)
Data	(n bytes, where n = Data Length)
Checksum	(4 characters)

- Each of the 4 character fields contains a positive integer in the range 0 to 35,535. All values will be transmitted in ASCII string representing the HEX value of the number. Therefore the number 255 is transmitted as “00FF”. Each of the number fields must be exactly 4 characters long.
- The ID Number is any number that the client wishes to send to the server. The server will always set this number to be the same in any packets it returns to the client with regard to this initial packet. For example, if the Palm sends a “play” packet with ID Number “0F01”, then the MRx will respond with an ACK and an ID Number “0F01”.
- The fragment number is described below under “Fragmentation.”
- The Data field can be at most 1476 bytes long.
  This length was chosen to ensure the total packet size is less than 1500 bytes. This is because the Ethernet specification requires packets of length 1500 bytes or less, and IrDA requires 2048 bytes or less. Therefore the same packet is able to be transmitted with either protocol.
- The Data may be either simple text, or binary data. This will be packet type dependent. There may also be no data.
- The Checksum is a simple sum of the previous values, mod FFFF. To calculate, add the value of each previous byte in the packet (from the Packet Type through to the end of the Data fields), and then only keep the lower 32 bits (mod FFFF, or AND with FFFF).

6.1.2 Retransmission:

- If a packet is sent, and there is no acknowledgment for 5 seconds, the packet should be retransmitted.
- A packet should be retransmitted at most 3 times.
- Note that ACK packets need no acknowledgment.

6.1.3 Fragmentation:

- Fragmentation is needed for data transmissions that will not fit into a single packet. The following packet types may be fragmented:
  - Directory listing (MP3 or list of playlists as a directory listing)
  - Update Playlist (packet sent by client containing playlist contents)
  - Request for playlist contents (packet sent by server containing playlist contents)
  - MP3 file transfer (packet sent by server containing the MP3 file).
- The transmit/receive functions for each device should be able to handle fragmented packets, if it will ever transmitting or receiving a fragmented packet.
- It is up to the individual devices how to treat fragmented packets. The two options are:
  1. Reassemble the packet before processing it at a higher level
  2. Process each fragment of the packet as it arrives
- Each fragment of a fragmented packet will itself be a valid packet. Therefore it will have a valid data length (for that packet), and checksum (for that packet). It will also require any ACKs that are normally generated for that type of packet. All header values of the packet other than fragment number (i.e. type, command and ID) will be identical for all fragments of the original packet.
- To acknowledge a fragmented packet, the ACK packet will have a fragment number indicating which fragment is being acknowledged.
- The client may return a NACK to a fragmented packet to indicate that no further fragments (of the transmission) should be transmitted. This is useful for cancelling a transfer of an MP3, or a very large play list if it is taking too long.
- If a packet has not been fragmented, then it's Fragment Number is 0.
- If a packet has to be fragmented, then the fragments are numbered in the following order: 1, 2, 3, … (up to at most 65535), 0.
  - Note that the final packet is always 0. This is true whether or not fragmentation is required.
- If a sender wishes to stop transmitting a set of fragments, it may simply stop sending them. For example, if the File Server is closed while sending an MP3 file to the MRx. The device receiving the fragmented packet should wait no more than 30 seconds for the next fragment of a packet before giving up. It is up to the receiving device to decide what to do with the fragments already received.

6.1.4 Message Types:

Name	Value	Description
MSGTYPE_INVALID	0	Invalid. Catch any errors where the message type has not be explicitly set.
MSGTYPE_DEBUG	1	Debug packets. No commands taken.
	2 and 3	Reserved. Do not use.
MSTTYPE_FILE	4	File commands.
MSGTYPE_PLAYCONTROL	5	Play control commands
MSGTYPE_MESSAGECONTROL	6	ACK/NACK

6.1.5 Commands:

File Commands (for MSGTYPE_FILE)

Name	Value	Description
CMTTYPE_FILE_INVALID	0	Invalid. Used to catch unassigned values.
CMTTYPE_FILE_REQDIRECTORYLISTING	1	Request a directory listing
CMTTYPE_FILE_RETDIRECTORYLISTING	2	Return a directory listing
CMTTYPE_FILE_REQFILE	3	Request a file
CMTTYPE_FILE_RETFILE	4	Return a file
CMTTYPE_FILE_STOREPLAYLIST	5	Save a playlist
CMTTYPE_FILE_DELPLAYLIST	6	Delete a playlist
CMTTYPE_FILE_SEARCH	7	Request for a file search.

Play Control Commands (for MSGTYPE_PLAYCONTROL)

Name	Value	Description
CMDTYPE_PLAYCONTROL_INVALID	0	Invalid. Used to catch unassigned values.
CMDTYPE_PLAYCONTROL_PLAY	1	Play current song
CMDTYPE_PLAYCONTROL_PAUSE	2	Pause current song
CMDTYPE_PLAYCONTROL_STOP	3	Stop if playing
CMDTYPE_PLAYCONTROL_SKIPFORWARD	4	Skip to next song
CMDTYPE_PLAYCONTROL_SKIPBACKWARD	5	Skip to previous song
CMDTYPE_PLAYCONTROL_REWIND	6	Rewind a few seconds
CMDTYPE_PLAYCONTROL_FASTFORWARD	7	Fast forward a few seconds
CMDTYPE_PLAYCONTROL_REQSTATUS	8	Request a status update
CMDTYPE_PLAYCONTROL_RETSTATUS	9	Return a status update

Message Commands (for MSGTYPE_MESSAGECONTROL)

Name	Value	Description
CMDTYPE_MESSAGECONTROL_INVALID	0	Invalid. Used to catch unassigned values.
CMDTYPE_MESSAGECONTROL_ACK	1	Received a message and it passed the checksum check.
CMDTYPE_MESSAGECONTROL_NACK	2	Message was garbled, or used to terminate transmission of fragmented packets.
CMDTYPE_MESSAGECONTROL_ERROR	3	Indicate the operation failed.
CMDTYPE_MESSAGECONTROL_SUCCESS	4	Indicate the operation succeeded.

7. Test Plan

This section outlines some test plans we plan on using to ensure we have met all our requirements as detailed in this document. The test plans are also designed to assess the integrity of our unit.

7.1 Installation and Setup.

7.1.1 The user will plug the MRx into a network with a PC running Windows 9x/2000.
7.1.2 The user will install the Host software onto the PC.
7.1.3 The user will install the Palm Pilot software onto the Palm.
7.1.4 The user will configure the MRx to correctly communicate with the Host PC over the Ethernet.

7.2 PC Playlist Editing:

7.2.1 The PC control application will show a list of available playlists.
7.2.2 The user will select a playlist, and edit it.
7.2.3 A list of songs currently in the playlist will be displayed
7.2.4 The user will browse the directories and add a song to the playlist.
7.2.5 The user will save the playlist.

7.3 MRx User Interface

7.3.1 The MRx will display a list of available playlists
7.3.2 The user will select one of the playlists and start it playing.
7.3.3 The LCD will indicate that the playlist is playing.
7.3.4 The user will skip a song on the playlist.
7.3.5 The LCD will display the information about the new song that is playing.

7.4 Palm Pilot

(Assume that a playlist is currently playing.)
7.4.1 The Palm Pilot will display the current status of the MRx.
7.4.2 The user will command the MRx through the Palm Pilot to skip to the next song.
7.4.3 The Palm Pilot will indicate the new song that is playing.
7.4.4 The user will command it to stop playing.
7.4.5 The Palm Pilot will indicate that the unit has stopped.

8. Acknowledgements

Bandwidth-Unlimited would like to thank a number of persons and companies for helping us get our project moving, and keeping it going over the past five months. Dave Miller of DFM Technologies Inc. (and a friend of one group member) has been instrumental in acquiring hardware and software support for the main platform for the MRx. He has also helped us choose hardware and software components and provided support in setting up the software. Dave introduced us to Insight Components who loaned us the Cirrus Logic evaluation board (worth $1500US). In particular, Mehdi Tamehi and Doug Stewart of Insight Components have worked with us to loan us the board and the software needed to run on it. The operating system running on the board is LynxWorks BlueCat 3.1. We would like to thank Ron Lockard of LynxWorks for the direct support he has given to us to configure and use BlueCat Linux. Also, we would like to thank Nels Esterby (local Cirrus Logic representative) of Micro-Electronics for providing assistance in borrowing the evaluation board from Insight Components.

For hardware component support, we would like to thank Dave Easingwood of Insight Components Inc for supplying us with some of the crucial components required for the board and very helpful support. Finally, ON Semicoductor Inc. supplied us with several peripheral components for the board. Our heart felt thanks to everyone for making our project possible.

9. Conclusion

The MP3 market is an exciting and rapidly expanding area in today's digital world. The numerous easily obtainable file-sharing programs allowed people to accumulate massive archives of MP3 encoded music. Until now, users have been restricted to listening this music on their computers or on headphones of their portable MP3 players. Bandwidth Unlimited will create the MRx Home Theatre Interface, to bring these massive MP3 libraries into any room of the house for enjoyment on real, full-sized home theatre systems.

The system consists of a host computer containing the music library, the MRx device receiving MP3 encoded audio via an Ethernet connection and a Palm Pilot remote control to provide a user-friendly interface to the MRx. The implementations for these features as described in this document were carefully planned and designed for reliability as well as usability. Developing this project according to these design specifications will certainly prove to be a challenging and rewarding experience.