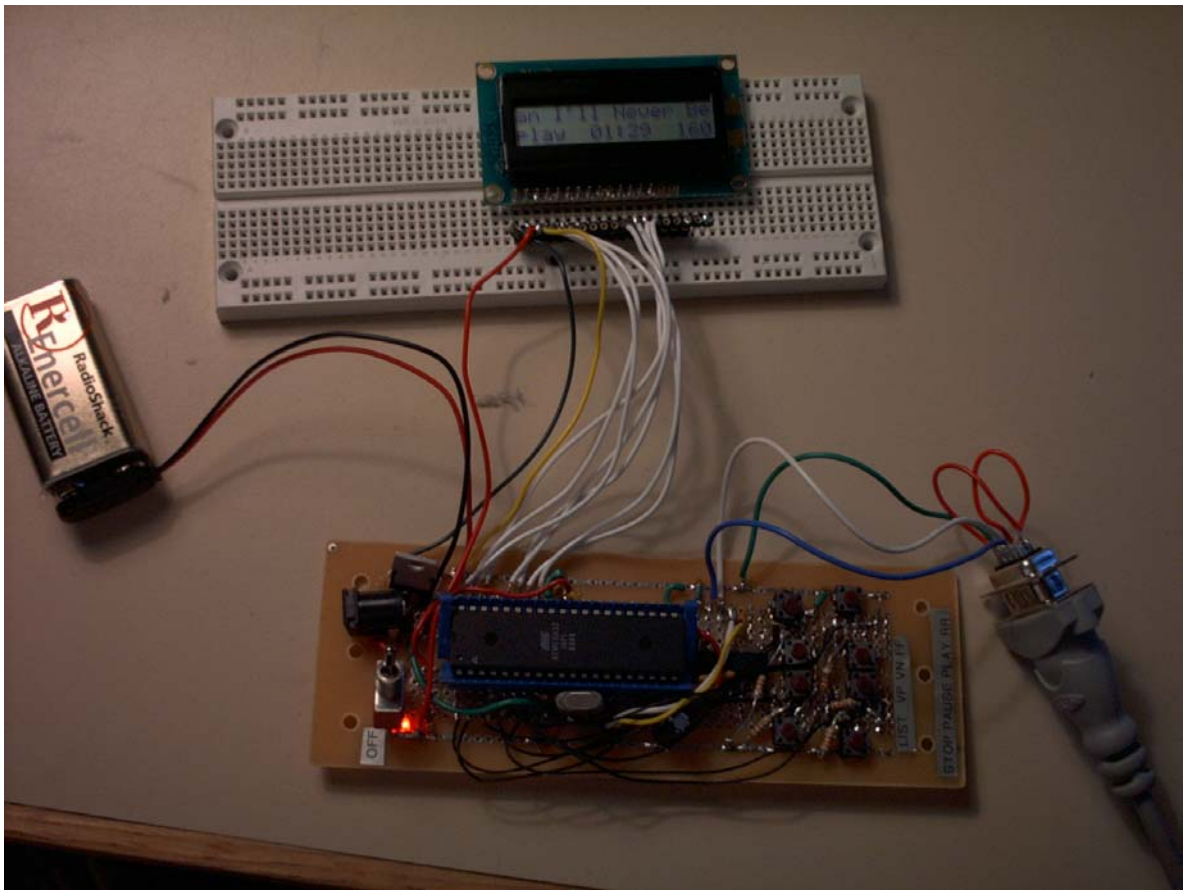
WINAMP REMOTE CONTROL

ECE-476

Joe Golden – jg325
Kenny McNutt – km279

05/06/04



INTRODUCTION

Our goal was to create a remote control for the popular music player, Winamp.

The Winamp controller is a standalone piece of hardware that allows a user to view playlists and control Winamp as if they were actually at the computer. It utilizes Winamp's API through a simple plugin that is loaded upon starting the application. The remote consists of eight buttons, each of which controls various parts of Winamp. The built in LCD provides feedback of the current song, play status, and allows users to view the current playlist.

HIGH LEVEL DESIGN

Rationale

Winamp has been one of the most popular music players since it began in 1997. With the users in place, who would not want to have a standalone remote to control Winamp?

Background

I have wanted to build a WinAmp remote control for a long time now. Several years ago I found a plugin that allowed the user to control the basic functionality of WinAmp with a few pushbuttons over serial communications. From that point I figured the next logical step was to provide feedback to the user via an LCD. That is when the idea was born. I had neither the time nor effort available until we were required to build something for ECE 476. I approached Joe with the idea, and aside from him wanting to build a self contained air hockey puck, he was all for the idea... and the WinAmp Controller was born.

Logical Structure

The Winamp controller can be split into two basic pieces: the plugin and the controller, which communicate serially.

Plugin
The plugin is a way to hook into WinAamp. It is also known as a dynamic linked library or dll. It is nothing more than a backend server waiting on requests via the com port. When a request is received (a byte comes in on the receive line), the data is translated and the request is serviced either by sending a command to WinAmp, or returning some data regarding the request.

Controller
The controller provides the interface for the user, consisting of eight buttons and an LCD. The hardware runs on a 16.0 MHz Atmel Mega 32 microcontroller.

Hardware/Software Tradeoffs
As a first step we wrote an external application to control WinAmp. The drawbacks of this implementation were lack of overall control of WinAmp, and the application had to be started separate

from, and after WinAmp had started. The external application limited the control to simply play, pause, stop, volume, and track changes.
Since our goal was to provide the user with a more powerful interface, including viewing current song information and the entire playlist, we had to develop a plugin. Now we have total control of WinAmp's API and the plugin loads when WinAmp is started.

HARDWARE/SOFTWARE DESIGN

Software

Plugin
Most of this project consisted of software. On the Windows side, we had to develop a plugin to tie into WinAmp. Although the end product is not complicated, starting from scratch was very difficult. Before this project, we knew nothing about dynamic link libraries or window handlers. The documentation for WinAmp's API is scarce. They just revamped the site for WinAmp v.5 so much of the past documentation was not available. After scouring to the ends of google, we were able to put several ideas together and create a successful plugin by spawning a new thread from within the init function of the plugin.

Communication between the Atmel and plugin is taken care of by a free communications library provided by BBDSoft. The Atmel sends all requests as single bytes. The plugin interprets the byte, and responds by either sending a command to Winamp, or returning information to the Atmel. Here is the basic structure of the communications protocol for the plugin.

byte Rx	function	returns	format of return
0	clock time	time	"dddd mmm dd hh:mm:ss yyyy"\v
1	play	n/a	n/a
2	stop	n/a	n/a
3	pause	n/a	n/a
4	volume up	n/a	n/a
5	volume down	n/a	n/a
6	nothing	n/a	n/a
7	play status	play,pause,stop	"status"\v
8	sample rate	sample rate	“samplerate”\v
9	bitrate	bitrate	“bitrate”\v
a	channels	# channels	“channels”\v
b	next track	n/a	n/a
c	prev track	n/a	n/a
d	fforward	n/a	n/a
e	rewind	n/a	n/a
f	version	version #	WinAmp "version"\v
g	play time	play time	"mm":"ss"\v
h	song title	title	"title"\v
i	entire playlist	playlist	"song1"\r"song2"\r…\v
j	playlist, 2 songs(incrementing)	2 titles	"*song1[0:14] song2[0:14]"\v
k	playlist, 2 songs(decrementing)	2 titles	"*song1[0:14] song2[0:14]"\v
l	playlist, 2 songs	2 titles	"*song1[0:14] song2[0:14]"\v
m	play current playlist selection	n/a	n/a

The quotes in the “format of return” column signify a variable. Brackets indicate byte 0 to 14 of the song titles. All returns end with a vertical tab, ‘\v’, to inform the Atmel the end of data. It was decided to use the vertical tab because of the improbability of its use in any other sense. We considered using the new line character, ‘\n’, but wanted to reserve it for future use.

Atmel

The ATMEL code is built around polling for button pushes using the Choice State Machine. This state machine runs in the main while(1) loop as fast as possible. When a button is pushed, the value is decoded and the correct state is entered on the next execution loop. Upon entering one of the new states, the button Debounce State Machine is called every 30ms. The Play, Pause, Stop, and mode states return to the Choice state once the button is debounced and the correct byte is sent to the UART. The other states, RWD, FF, and Volume Up and Down, have an added feature. If the user presses the button and releases in under 400ms, a single command is issued, writing a byte to the UART. If the button is held down, Pushflag=1, the volume up and volume down will begin to autoincrement once every 400ms.

The RWD and FF will rewind and fast-forward instead of skip to the previous or next song.

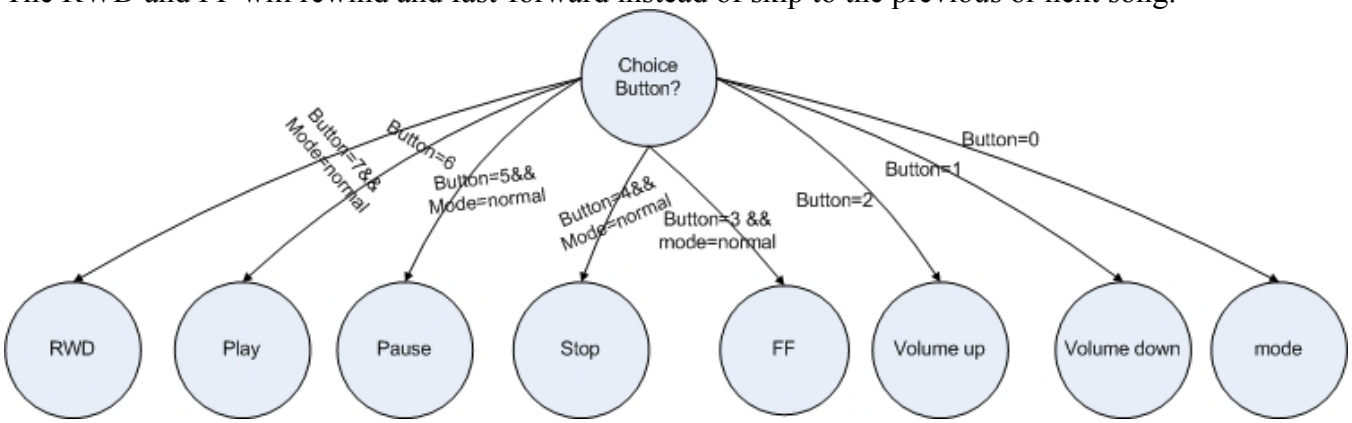


Table 1: Choice State Machine

The mode button changes the user from normal mode, viewing the current song information, to playlist mode, viewing the current playlist, viewing two songs at a time. When in the playlist mode, only the Play, Volume Up, Volume Down, and mode buttons are enabled. The mode button simply toggles the mode, and the volume buttons scroll the playlist up and down. The play button plays the currently highlighted song on the playlist. It is called every 30ms as needed.

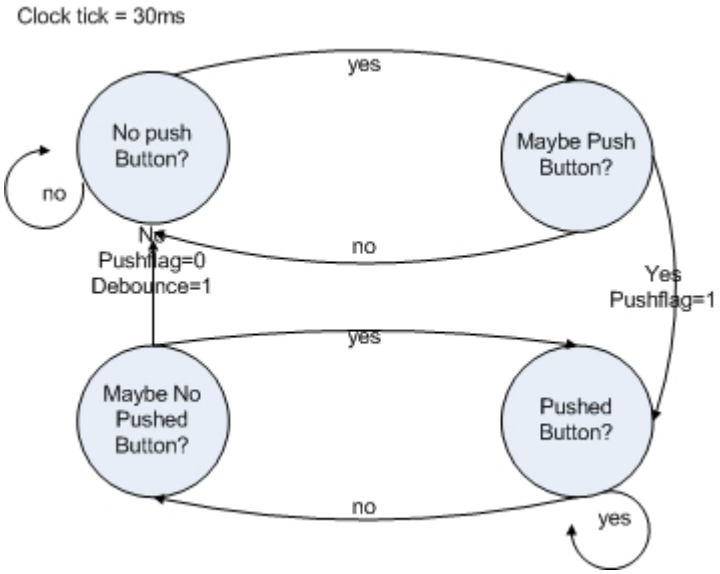


Table 2: Debounce State Machine

The Atmel uses interrupts to signal the main program when there is data to be read on the UART. The UART buffer is read until the vertical tab, '\v', character is read signifying end of transmission. Once a request is sent that requires a response, the UART receive interrupt is enabled and the Print State Machine stays in its current state until the data is ready, "cmd_ready=1".

The Print State Machine flows as follows. Every 50ms, the Print State Machine runs. The initial begin state is Play Status. This state sends a request to the plugin for the current status of the song. The plugin returns play, pause, or stop. Once the data is received, the LCD is updated with the new status and the state is then set to the Song Title state.

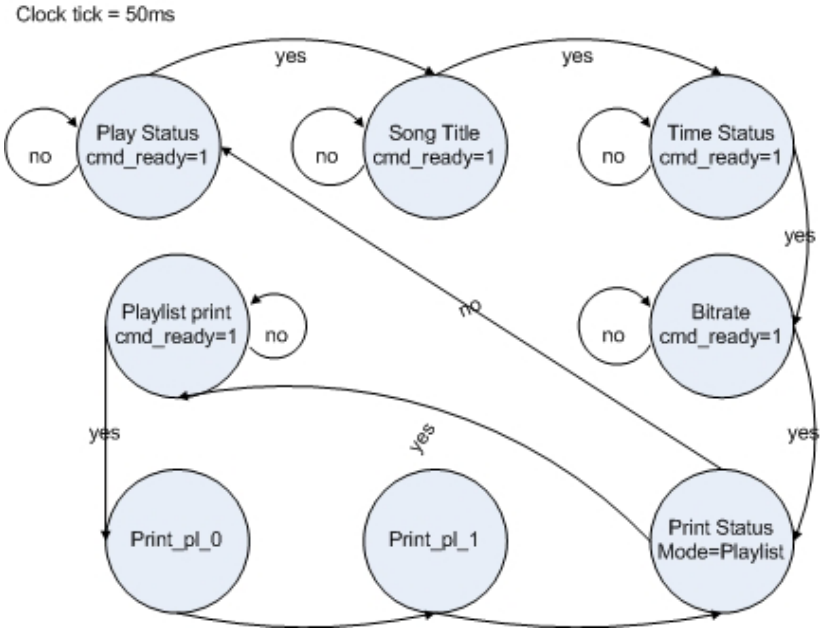


Table 3: Print State Machine

The Song Title state sends a request to the plugin for the song title information. The plugin returns the name of the song that is displayed in WinAmp's playlist. The format of this data can be changed from within WinAmp's preferences menu. Upon receiving the data, the LCD is updated with the song title and the state changes to Time Status.

Time Status and Bitrate are similar to the first two states. They send requests to the plugin and upon receiving the data, the LCD is updated accordingly. After Bitrate is complete, the state machine goes to the Print Status state. Here the new state is determined by the current mode. In this case the machine returns to the Play Status state yielding an overall LCD update in normal mode of 250ms plus the time to execute the button state machine.

When the controller is in playlist mode, set in the Choice State Machine's mode state, the state of the Print State Machine is set to Playlist Print state. This state retrieves the playlist and formats its. It then sets the current state to Print_pl_0 where the first line of the playlist is written to the LCD. The next state is Print_pl_1 where the second song of the playlist is written to the LCD. Once the playlist has been written to the LCD, the Print Status state determines whether to set the new state to Play Status or Print Playlist based on the current mode.

A key feature of using a state machine like the Print State Machine is the LCD is always up to date. If a user changes song information, play status, or even deletes a song in the playlist, the controller is almost immediately updated. To the user, it should seem to have very little lag.

Built into the controller is another feature known as a time-out timer. If the communications are ever lost, the timer will timeout in 500ms and re-initialize the entire controller. This is to prevent the controller from losing communications and potentially staying out of synch, never regaining synchronization.

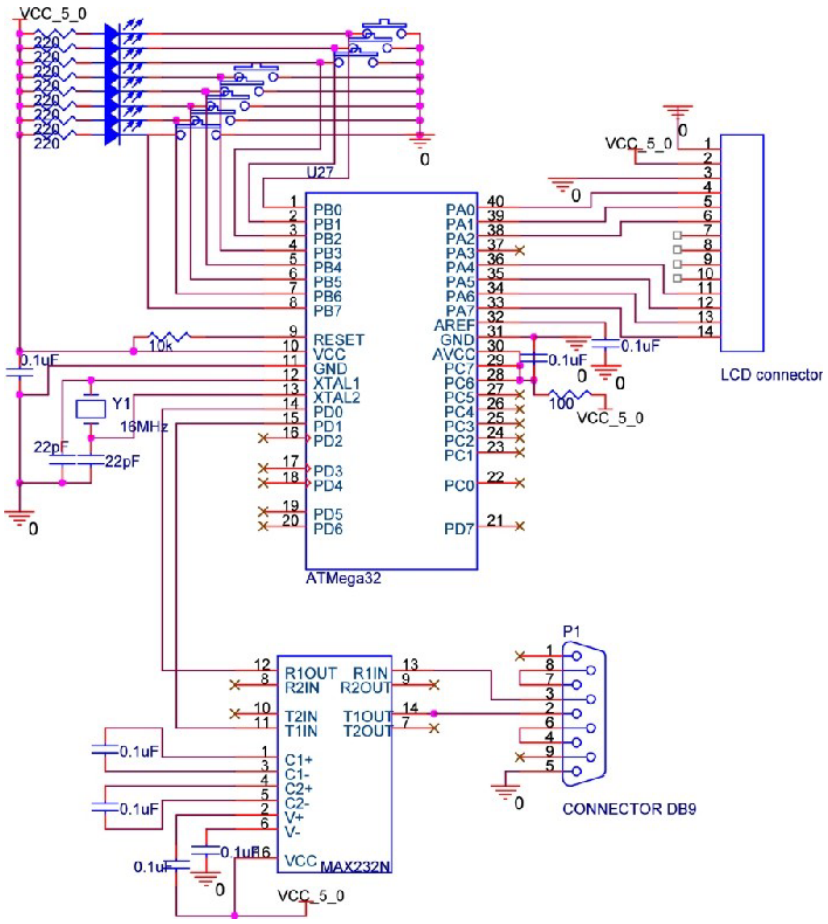
Hardware

For our project, we used an Atmel Mega32 micro controller that we ran at 16MHz. The circuit diagram can be found in the Schematic section. This Atmel has four I/O ports. Port A was used to output to the LCD. Port B is an input from the pushbuttons that ultimately sends commands to WinAmp. Port C was used as a test and debugging port during development. We also used two pins on Port D for serial communication.

Our 5V power was supplied from a 9V battery that was regulated down by using an LM340 5V regulator. We also placed a 100uF capacitor on the output of the regulator to help smooth out the signal. Aside from this, given the schematic hardware schematic, this project should be easily duplicated.

We experienced two problems with the hardware. First, we mistakenly assumed that the LEDs had a 0.7V drop across them. Originally, the LEDs were placed on the other side of the button since a 0.7V signal would be considered a low signal by the micro controller anyway. However, we soon learned that yellow LEDs require about 2.6 volts to operate. This obviously caused problems because we never could give a low input. The second problem we faced was that the input pins seemed to do nothing. The micro worked fine on the STK500 with the entire circuit set up. We then put it on the stand alone board we made and it did not work. So, to see if the micro and clock were working, we wrote a quick program that just changed all of port C from high to low every two seconds. This worked fine. We knew that it was not a hardware issue, but we could not figure out what could possibly be wrong with the code so that the micro would work on the STK but not on a standalone board. Finally, we remembered from lab 1 that the micro controller would not work without the LCD connected to it, which we had failed to disconnect from the STK and connect to the stand-alone board. Once we realized this, our board worked very well.

Schematic



RESULTS OF DESIGN

We feel our design was very simple and elegant. Response time from the button to execution of the command is almost instantaneous except for the read play list function. When scrolling up and down the play list, a slight delay can be seen. This is the only function that needs a button push to begin the command, and then must wait for a response from the computer. Because of this, there is a delay of approximately one quarter of a second.

Accuracy is very good on our project. We never pushed a button and had a wrong command. The only problem ever experienced was with our fast forward and rewind commands. It is set up so that if the fast forward button is pressed and released, it skips to a new song. If it is pressed and held, it fast forwards through the current song. The same is true for rewind. There were a few times when we wanted to skip to the new song and we ended up fast forwarding through the current song instead.

Safety was not much of a concern during our project. The 9V battery can only be connected the correct way. Our buttons were clearly labeled. If going into production, there would be real PCBs made and a case for the controller that would eliminate safety issues of exposed wires.

Our project does not interfere to our knowledge of any other design projects. The only other possible interference would be if someone was using WinAmp on the computer while someone else was using the remote control. Because our code is non-blocking of normal use, the two people can both control it at the same time, which could lead to fights about which song to play.

Known Bugs

As much as we would like to say there are none, there are some bugs. Inevitably with any software, there will be bugs, and more will continually creep up.

- If the playlist mode on WinAmp is set to shuffle, and the playlist mode on the contoller is turned on, this will cause WinAmp to crash.
- If the last song in the playlist is deleted, and playlist mode is entered from the controller, WinAmp will crash.
- Occassionaly the LCD will bug out when some other action on the computer takes place.
- At some point, Mozilla crashed, claiming there was a bug in the com port software that the plugin uses to service requests.

USER'S MANUAL

Step 1: Start WinAmp on the computer and set up a playlist. Load the plugin file that comes with the remote control onto the computer. Simply find the WinAmp directory on the hard drive, and place the plugin file given into the plugin directory. This only needs to be down the first time you use the remote control.

Step 2: Power on the remote control by flipping the toggle switch. A red LED should illuminate.

Step 3: All buttons are labeled accordingly on the remote control but they are listed here with functionality. When a button is pressed, a yellow LED should illuminate next to the button.

Button 0: MODE: When pressed, this allows the user to see the playlist. Use the volume buttons to scroll up and down the list. A * is also visible on the left side of the screen. To play a song, simply press the PLAY button when the * is next to the chosen song. To exit from play list mode, hit the PLAYLIST button again.

Button 1: VOUME DOWN: This button decreases the volume. However, when viewing the play list, it allows the user to scroll down. When the button is held down, it will decrease the volume at a faster pace.

Button 2: VOLUME UP: This button increases the volume. However, when viewing the play list, it allows the user to scroll up. When the button is held down, it will increase the volume at a faster pace.

Button 3: FAST FORWARD: When tapped, this button skips to the next song on the play list. When held down, it will fast forward through the current song.

Button 4: STOP: This button stops music from playing.

Button 5: PAUSE: This button will pause a song at its current location. To exit PAUSE, hit either the PAUSE or PLAY button.

Button 6: PLAY: This button will play the first song on the play list if it is the first button pressed. If music had already been playing, it will play whichever song the music was stopped at. When pressed while viewing the play list, the song with the * next to it will begin to play.

Button 7: REWIND: When tapped, this button skips to the previous song on the play list. When held down, it will rewind through the current song.

Frequently Asked Question

The LCD is small, what if two songs have the same first 16 characters?
The remote control sets up a scrolling song title. This means the entire song name will scroll across the LCD

What is shown on the LCD during normal operation?
You will see the current song being scrolled across the top line. On the bottom line, it shows what the current state of operation is. If the state is PLAY, the bottom line will also show how much time has passed in the current song as well as the current bit rate.

What kind of batteries does it use?
One 9V battery will last for several hours of usage.

What does it mean if the LCD reads "no connection"?
There can be one of several problems. WinAmp could be closed, the plugin might not be installed, or the serial cable could be loose or not plugged into com 1. As a first step, trying resetting the controller and restarting WinAmp.

COST

Part	Quantity	Estimated Cost	Cost
Atmel Mega32 CPU	1	$8.00	$8.00
Switch	8	$1.80	$1.80
LM340 voltage regulator	1	$1.65	$1.65
*16MHz crystal	1	$0.62	$0.00
*220 resistor	8	$0.32	$0.00
*100 resistor	1	$0.04	$0.00
*10k resistor	1	$0.04	$0.00
*100uF capacitor	1	$0.15	$0.00
*MAX232N	1	$0.78	$0.00
9V battery	1	$2.00	$2.13
*0.1uF capacitor	8	$1.20	$0.00
*22pF capacitors	2	$0.14	$0.00
*40 pin socket	1	$0.00	$0.00
DB9 connector	1	$1.00	$1.30
solder board	1	$2.50	$2.50
LEDs	9	$1.00	$1.53
*Serial Cable	1	$0.00	$0.00
LCD	1	$5.00	$5.00
Total		**$26.24**	**$23.91**

Table 4: *These parts were borrowed, donated, or otherwise acquired.

CONCLUSION

The hardware design for this project was fairly simple and straightforward. There would not be much to do differently. It could obviously be made smaller with the use of a PCB and surface mount components, but for a prototype it works well

The software design for this project was a little overwhelming at times. It is not difficult to create a plugin for WinAmp, but when you begin with absolutely no knowledge of plugins or Windows style programming, it can be a lot to swallow. We have noticed the occasional strange event occur in Windows. If there was more time, it would be nice to do more debugging and see what kinds of effects various applications have on the plugin.

The only applicable standard to our design is the RS232 standard. The following is an excerpt from "The RS232 Standard" by Christopher Strangio.

> In the 1960's, a common interface standard for data communications was developed. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel

> require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

To comply with this standard, we simply needed to add a MAX232 chip to help convert voltages to and from the micro controller to the serial connection in order to comply with the standard.

There were several intellectual property considerations for our project. For serial communications we used a free general component library for WIN32 from BBDSoft. They can be found via the web at http://www.bbdsoft.com. Also, the main part of the project was the WinAmp API. WinAmp has a large group of private developers constantly creating new skins, plugins, and visiualizers. All the API headers are free of charge as is the software. The Debounce State Machine was presented to the class by Professor Land.

Ethically, we followed to the best of our knowledge, all ten points in the IEEE Code of Ethics both in the design and in this report. We have given credit where credit is necessary. We did not take bribes. We worked to advance our understanding of a subject matter. We did not discriminate nor did we place anyone in harms way at any point during this project.

If working to advance this project, there is still plenty of work to be done. First thing would be to get a wireless system at a reasonable price. We would need two transceivers for this project and good transceivers cost a minimum of $20 each. What distinguishes our project from a simple remote control that is available commercially is that we have a play list function that can display on the LCD. Without this, only one transmitter and one receiver would be necessary and this would tremendously cut costs since a transmitter and receiver together can be found for $10. It should also be noted that by placing a small hard drive or flash card into the remote control, it would be easy to convert it to a portable MP3 player. Other improvements that can be made would be to make a nice case for the board, and to reduce its size through a PCB and surface mount components. By doing this, the board size can easily be cut in half. Another improvement would be a much larger LCD. This would allow more information to be seen. Finally, the system can be made more robust. Finally, certain bugs can be worked out and improved on. For instance, when WinAmp is set on “shuffle” mode, our controller has major problems when trying to view the play list. Bugs like this, and other bugs found through testing, can be worked out and fixed.

We do feel there would be a market for our product if cheap transceivers can be found. In particular, we feel the college student would be our target audience. While at parties, it is common for the host to set up a big play list so that music is taken care of all night. The host would not want people entering his bedroom at will to change the song and traditional remote controls would make this necessary since they require line of sight. The host would also not want to leave the party to constantly go to his bedroom to make changes. Our remote control would enable someone, from the living room, to control the play list without entering private areas of the house.

RESOURCES

1. MAX232 Data Sheet -
   http://rocky.digikey.com/WebLib/Texas Instruments/Web data/MAX232,232I.pdf
2. Atmel Mega32 Data Sheet – instruct1.cit.cornell.edu/courses/ee476/AtmelStuff/full32.pdf
3. Serial Communications Library – www.bbdsoft.com
4. WinAmp API and Application – www.winamp.com

CONTRIBUTIONS OF EACH PARTNER

Joe Golden - jg325@cornell.edu
Designed and built the hardware.

Kenny McNutt - km279@cornell.edu
Wrote the plugin, Atmel code, and designed the webpage.

APPENDIX – Source Code

remote_plugin.cpp

```
#include "stdafx.h"
#include "windows.h"
#include "wa_ipc.h"
#include "gen.h"
#include "winamp.h"
#include <process.h>
#include <iostream>


#include "COMPORT.H"
#include <string.h>
#include <sys/timeb.h>
#include <time.h>

#define SAMPLERATE 0
#define BITRATE 1
#define CHANNELS 2
#define SECONDS 1
#define MSECONDS 0

int init();
void config();
void quit();
static void Reader( void* );
int play(void);
void volumeUp(void);
void volumeDown(void);
void stop(void);
void pause(void);
int getStatus(void);
int getInfo(int);
void nextTrack(void);
void prevTrack(void);
void fforward(void);
void rewind(void);
int stringSize(char *);
int getVersion(void);
int getPlayTime(void);
int getPlaylistIndex(void);
int getPlaylistLength(void);
char* getPlaylist(void);
char * remove_nulls(char[], int);

/************************************************************/
/************winamp plugin stuff*********************************/
/************************************************************/

winampGeneralPurposePlugin plugin =
{
        GPPHDR_VER,
        "Winamp Remote v1.0",
        init,
        config,
        quit,
};
//required function
int init()
{
        //spawn new thread for main servicing function
        _beginthread( Reader, 0, NULL );
        return 0;
}
//required function
void config()
{
```

```
}
//required function
void quit()
{
}

extern "C" __declspec( dllexport ) winampGeneralPurposePlugin * winampGetGeneralPurposePlugin()
{
	return &plugin;
}

/*****************************************************************/
/************end winamp plugin stuff*****************************/
/*****************************************************************/


//function that Rx, Tx, and communicates with winamp
static void Reader( void* )
{
	HWND mainwinampwin = plugin.hwndParent;
	int status;
	char choice;
	int version;
	char aData = 'h';
	double x = clock();
	CString trackFilename;
	struct _timeb timebuffer;
	char *timeline;
	char *title;
	int pos;
	int current_pos=0, song_pos;
	char buffer[10];
	char plbuffer[100];
	int i;
	char clear_flag=0;

	/************************************************/
	/******setup com port*****************************/
	/************************************************/
	COMPort aPort ("COM1");                                  //com1
	aPort.setBitRate (COMPort::br9600); //9600bps
	aPort.setParity (COMPort::None);      //no parity
	aPort.setDataBits (COMPort::db8);     //8 data bits
	aPort.setStopBits (COMPort::sb1);     //1 stop bit
	aPort.setxONxOFF (false);                    //no xOnxOff
	aPort.setHandshaking(false);          //no handshaking
	/************************************************/
	/******end setup com port*************************/
	/************************************************/

	char *output;

  while(1 )
  {
		//block until remote sends a single byte of data
		choice = aPort.read();

		//all data that is transmitted ends with a vertical tab '\v'
		//I figure this is not used in any text so it makes a good terminator
		switch (choice)
		{
		//current time
		case '0':
			//*******************************
			//   get time
			_ftime( &timebuffer );
			timeline = ctime( & ( timebuffer.time ) );
			aPort.write(timeline, stringSize(timeline));
			aPort.write("\v",stringSize("\v"));
			//*************************************
			break;
```

```
//play
case '1':
	play();
	break;
//stop
case '2':
	stop();
	break;
//pause
case '3':
	pause();
	break;
//increase volume
case '4':
	volumeUp();
	break;
//decrease volume
case '5':
	volumeDown();
	break;
//nothing
case '6':
	break;
//get status:  play, pause, stop
case '7':
	status = getStatus();
	if(status == 0)
		aPort.write("stop\v",stringSize("stop\v"));
	else if (status ==1)
		aPort.write("play\v",stringSize("play\v"));
	else if (status == 3)
		aPort.write("pause\v",stringSize("pause\v"));
	else
		aPort.write("unknown\v",stringSize("unknown\v"));
	break;
//samplerate
case '8':
	//format the output
	sprintf(buffer,"%d\v",getInfo(SAMPLERATE));
	aPort.write(buffer,stringSize(buffer));
	break;
//bitrate
case '9':
	//format the output
	sprintf(buffer,"%d\v",getInfo(BITRATE));
	aPort.write(buffer,stringSize(buffer));
	break;
//number of channels
case 'a':
	//format the output
	sprintf(buffer,"%d\v",getInfo(CHANNELS));
	aPort.write(buffer,stringSize(buffer));
	break;
//go to next track
case 'b':
	nextTrack();
	break;
//go to prev track
case 'c':
	prevTrack();
	break;
//fast forward
case 'd':
	fforward();
	break;
//rewind
case 'e':
	rewind();
	break;
//get version of winamp
case 'f':
```

```
                        version = getVersion();
                        //format the output
                        sprintf(buffer,"WinAmp v%x.%02x\v",version/4096, version&0xFFF);
                        aPort.write(buffer,stringSize(buffer));
                        break;
                //get current play time
                case 'g':
                        //format the output
                        sprintf(buffer,"%02d:%02d\v",getPlayTime()/1000/60,(getPlayTime()/1000)%60);
                        aPort.write(buffer,stringSize(buffer));
                        break;
                //get title of current playing song
                case 'h':

        title=(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,getPlaylistIndex(),IPC_GETPLAYLISTTITLE);
                        //format the output
                        sprintf(buffer,"%s\v",title);
                        aPort.write(buffer,stringSize(buffer));
                        break;
                //get entire playlist
                case 'i':
                        sprintf(plbuffer,"");
                        pos=1;
                        while(pos < getPlaylistLength())
                        {
                                title =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,pos,IPC_GETPLAYLISTTITLE);
                                //format the output
                                //put a \r between songs in the playlist
                                sprintf(plbuffer,"%s%s\r",plbuffer,title);
                                pos ++;
                        }
                        //add the vertical tab as an end of data character
                        sprintf(plbuffer,"%s\v",plbuffer);
                        aPort.write(plbuffer,stringSize(plbuffer));
                        break;
                //get only two songs to display in playlist(incrementing)
                case 'j':
                        if(current_pos+1 < getPlaylistLength())
                        {
                                current_pos++;
                        }

                        title = (char*)SendMessage(plugin.hwndParent,WM_WA_IPC,current_pos-
1,IPC_GETPLAYLISTTITLE);

                        //fill first 16 characters of buffer with first song
                        //add a space to beginning
                        plbuffer[0]=' ';
                        for(i=1;i<16;i++)
                        {
                                plbuffer[i]=title[i-1];
                        }

                        //remove the null terminating character and add blank spaces
                        //after the null is found, replace it and all remaining with spaces
                        for(i=1;i<16;i++)
                        {
                                if(plbuffer[i]==0)
                                {
                                        plbuffer[i]=' ';
                                        clear_flag=1;
                                }
                                else if(clear_flag==1)
                                {
                                        plbuffer[i]=' ';
                                }
                        }
                        clear_flag=0;

                        //get second song to send
```

```
                        if(current_pos < getPlaylistLength())
                        {
                                title =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,current_pos,IPC_GETPLAYLISTTITLE);
                                //fill first 16 characters of buffer with second song
                                //add a star to beginning to signify current selected song in playlist
                                plbuffer[16]='*';
                                for(i=1;i<16;i++)
                                {
                                        plbuffer[i+16]=title[i-1];
                                }
                        }

                        //remove the null terminating character and add blank spaces
                        //after the null is found, replace it and all remaining with spaces
                        for(i=17;i<32;i++)
                        {
                                if(plbuffer[i]==0)
                                {
                                        plbuffer[i]=' ';
                                        clear_flag=1;
                                }
                                else if(clear_flag==1)
                                {
                                        plbuffer[i]=' ';
                                }
                        }
                        clear_flag=0;

                        //add a vertical tab and null terminate it
                        plbuffer[32]='\v';
                        plbuffer[33]=0;

                        //write to uart
                        aPort.write(plbuffer,stringSize(plbuffer));
                        break;
                //get only two songs to display in playlist(decrementing)
                case 'k':
                        if(current_pos-1 >= 0)
                        {
                                current_pos--;
                        }

                        title =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,current_pos,IPC_GETPLAYLISTTITLE);

                        //fill first 16 characters of buffer with first song
                        //add a star to beginning to signify current playlist selection
                        plbuffer[0]='*';
                        for(i=1;i<16;i++)
                        {
                                plbuffer[i]=title[i-1];
                        }

                        //remove the null terminating character and add blank spaces
                        //after the null is found, replace it and all remaining with spaces
                        for(i=0;i<16;i++)
                        {
                                if(plbuffer[i]==0)
                                {
                                        plbuffer[i]=' ';
                                        clear_flag=1;
                                }
                                else if(clear_flag==1)
                                {
                                        plbuffer[i]=' ';
                                }
                        }
                        clear_flag=0;
```

```
                        title =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,current_pos+1,IPC_GETPLAYLISTTITLE);

                        //fill first 16 characters of buffer with second song
                        //add a space to beginning
                        plbuffer[16]=' ';
                        for(i=1;i<16;i++)
                        {
                                plbuffer[i+16]=title[i-1];
                        }

                        //remove the null terminating character and add blank spaces
                        //after the null is found, replace it and all remaining with spaces
                        for(i=17;i<32;i++)
                        {
                                if(plbuffer[i]==0)
                                {
                                        plbuffer[i]=' ';
                                        clear_flag=1;
                                }
                                else if(clear_flag==1)
                                {
                                        plbuffer[i]=' ';

                                }

                        }
                        clear_flag=0;

                        //add a vertical tab and null terminate it
                        plbuffer[32]='\v';
                        plbuffer[33]=0;
                        aPort.write(plbuffer,stringSize(plbuffer));
                        break;
                //get the current two songs in playlist
                case 'l':

                        if(current_pos==0)
                        {
                                song_pos=current_pos+1;
                        }
                        else song_pos=current_pos;
                        title = (char*)SendMessage(plugin.hwndParent,WM_WA_IPC,song_pos-
1,IPC_GETPLAYLISTTITLE);

                        //add star to beginning of song to signify current playlist selection
                        if(song_pos-1==current_pos)
                        {
                                plbuffer[0]='*';
                        }
                        //add space at beginning of song
                        else plbuffer[0]=' ';

                        //fill buffer with first 16 chars of first song
                        for(i=1;i<16;i++)
                        {
                                plbuffer[i]=title[i-1];
                        }

                        //remove all null chars and fill remaining with spaces
                        for(i=1;i<16;i++)
                        {
                                if(plbuffer[i]==0)
                                {
                                        plbuffer[i]=' ';
                                        clear_flag=1;
                                }
                                else if(clear_flag==1)
                                {
                                        plbuffer[i]=' ';
```

```
                        }

                }
                clear_flag=0;


                if(song_pos < getPlaylistLength())
                {
                        title =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,song_pos,IPC_GETPLAYLISTTITLE);
                        if(song_pos==current_pos)
                        {
                                plbuffer[16]='*';
                        }
                        else plbuffer[16]=' ';
                        for(i=1;i<16;i++)
                        {
                                plbuffer[i+16]=title[i-1];
                        }
                }

                //fill with first 16 chars of second song
                for(i=17;i<32;i++)
                {
                        if(plbuffer[i]==0)
                        {
                                plbuffer[i]=' ';
                                clear_flag=1;
                        }
                        else if(clear_flag==1)
                        {
                                plbuffer[i]=' ';
                        }
                }
                clear_flag=0;
                //add vertical tab and null char
                plbuffer[32]='\v';
                plbuffer[33]=0;
                aPort.write(plbuffer,stringSize(plbuffer));
                break;
        //play current playlist selection
        case 'm':
                //set playlist position
                SendMessage(plugin.hwndParent,WM_WA_IPC,current_pos,IPC_SETPLAYLISTPOS);
                play();
                break;

        }

  }
}

/*****************************************************************/
/*************basic control functions*************************/
/*****************************************************************/
void volumeUp(void)
{
        SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_VOLUMEUP,0);
}

void volumeDown(void)
{
        SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_VOLUMEDOWN,0);
}

int play(void)
{
        SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_PLAY,0);
        return 0;
}
```

```
void stop(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_STOP,0);
}


void pause(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_PAUSE,0);
}

void nextTrack(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_NEXT,0);
}

void prevTrack(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_PREV,0);
}

void fforward(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_FF,0);
}

void rewind(void)
{
	SendMessage(plugin.hwndParent, WM_COMMAND,WINAMP_REW,0);
}
/*****************************************************************/
/**************end basic control functions***********************/
/*****************************************************************/

//get play status:  play, pause, stop
int getStatus(void)
{
	return SendMessage(plugin.hwndParent,WM_WA_IPC,0,IPC_ISPLAYING);
}

//get basic info : bitrate, samplrate, channels
int getInfo(int mode)
{
	return SendMessage(plugin.hwndParent,WM_WA_IPC,mode,IPC_GETINFO);
}

//get winamp version
int getVersion(void)
{
	return SendMessage(plugin.hwndParent,WM_WA_IPC,0,IPC_GETVERSION);
}

//current time elapsed of song
//time in seconds does not work.  Use MSECONDS.
int getPlayTime(void)
{
	return SendMessage(plugin.hwndParent,WM_WA_IPC,MSECONDS,IPC_GETOUTPUTTIME);
}

//returns the index of the song in the playlist that is currently playing
int getPlaylistIndex(void)
{
	return SendMessage(plugin.hwndParent,WM_WA_IPC,0,IPC_GETLISTPOS);
}

//returns pointer to entire playlist
char* getPlaylist(void)
{
	char* song;
	int pos = 0;
	char* buffer = "";
```

```
        while(pos < getPlaylistLength())
        {
                song =
(char*)SendMessage(plugin.hwndParent,WM_WA_IPC,getPlaylistIndex(),IPC_GETPLAYLISTTITLE);
                sprintf(buffer,"%s\r\n%s\r\n",buffer, song);
                pos ++;
        }
        return buffer;

}

//return the number of songs in the playlist
int getPlaylistLength(void)
{
        return SendMessage(plugin.hwndParent,WM_WA_IPC,0,IPC_GETLISTLENGTH);
}

//calculate the string size so i don't have to
int stringSize(char *inString)
{
        int x=0;
        while(inString[x]!=0)
        {
                x++;
        }
        return x;
}
```

wac_v1.c

```
#include <Mega32.h>

#include <stdio.h>
#include <delay.h>
#include <string.h>

#define LCDwidth 16
//port A
#asm
        .equ __lcd_port = 0x1B
#endasm

#include <lcd.h>

#define button0 0x01
#define button1 0x02
#define button2 0x04
#define button3 0x08
#define button4 0x10
#define button5 0x20
#define button6 0x40
#define button7 0x80

#define CHOICE 0
#define PLAY 1
#define STOP 2
#define PAUSE 3
#define VOLUMEUP 4
#define VOLUMEDOWN 5
#define RWD 6
#define FFWD 7
#define TOGGLE_MODE 8

#define NO_PUSH 0
#define MAYBE_PUSH 1
#define PUSHED 2
#define MAYBE_NO_PUSHED 3

#define PLAY_STATUS 0
#define SONG_TITLE  1
#define TIME_STATUS 2
#define PRINT_STATUS 3
#define PLAYLIST_PRINT 4
#define PRINT_PL_0 5
#define PRINT_PL_1 6
#define BITRATE 7

#define DOWN 0
#define UP 1
#define NONE 2

//mode defines
#define NORMAL         0
#define PLAYLIST       1

void debounce(int);
void init(void);
void toggleLED(unsigned int, char);
void checkStatus(void);
char * format(char[]);
unsigned char multi_space(char[]);

unsigned int pit_count;
unsigned char debounce_state;
unsigned char push_flag;
char debounced;
char adjust_done_flag;
char button_state;
```

```
char wait_for_input;
char status;
char c;
char cmd_str[250];
char song_title_full_prev[250];
char song_title_full[250];
char song_title_print[17];
char cmd_ready;
int char_count;
char play_status_str[6];
char lcd_buffer0[17];
char repeat_flag;
char mode;
char song_time[7];
unsigned char song_pos=0;
char pl_line0[17], pl_line1[17];
char bit_rate[6];
char output[12];
unsigned char counter, song_pos_timer;
unsigned int pl_pos;
unsigned int time_out_counter;
unsigned char start_counter;
unsigned char scroll;

//receive interrupt service routine
interrupt[USART_RXC] void uartRx(void)
{
    c = UDR;   //copy character from UDR to global C
       if(c!='\v') cmd_str[char_count++] = c;
       else
       {
              cmd_str[char_count] = 0;
              cmd_ready = 1;
              UCSRB.7 = 0;   //kill interrupt until software is ready
       }
}

//init Rx interrupt servicing
void get_cmd_init(void)
{
       char_count = 0;
       cmd_ready = 0;
       UCSRB.7 = 1;   //enable Rx interrupt
       time_out_counter=0;
       start_counter=1;
}


//main function
void main(void)
{
       int x;
       //init variables
       init();

       while(1)
       {

       //poll for button presses
       switch (button_state)
              {
                     case CHOICE:
                            if (~PINB == 0x0) break; //no button pushed
                            else if(~PINB == button6)
                            {
                                   toggleLED(button6, 'C');
                                   button_state = PLAY;
                            }
                            else if(~PINB == button5 && mode==NORMAL)
                            {
                                   toggleLED(button5, 'C');
```

```
                button_state = PAUSE;
        }
        else if(~PINB == button4 && mode==NORMAL)
        {
                toggleLED(button4, 'C');
                button_state = STOP;
        }
        else if(~PINB == button2)
        {
                toggleLED(button2, 'C');
                button_state = VOLUMEUP;
        }
        else if(~PINB == button1)
        {
                toggleLED(button1, 'C');
                button_state = VOLUMEDOWN;
        }
        else if(~PINB == button7 && mode==NORMAL)
        {
                toggleLED(button7, 'C');
                button_state = RWD;
        }
        else if(~PINB == button3 && mode==NORMAL)
        {
                toggleLED(button3, 'C');
                button_state = FFWD;
        }
        else if(~PINB == button0)
        {
                button_state = TOGGLE_MODE;
        }
break;

case PLAY:
        if (pit_count%30 == 0) debounce(button6);
        if (mode==NORMAL)
        {
                if (debounced)
                {
                        putchar('1');
                        button_state = CHOICE;
                        debounced = 0;
                        toggleLED(button6, 'C');
                }
        }
        else if (mode==PLAYLIST)
        {
                //play the current song viewed on the playlist
                if (debounced)
                {
                        putchar('m');
                        button_state = CHOICE;
                        debounced = 0;
                        mode = NORMAL;
                        toggleLED(button6, 'C');
                }
        }
        break;

case PAUSE:
        if (pit_count%30 == 0) debounce(button5);
        if (debounced)
        {
                putchar('3');
                button_state = CHOICE;
                debounced = 0;
                toggleLED(button5, 'C');
        }
        break;

case STOP:
```

```
        if (pit_count%30 == 0) debounce(button4);
        if (debounced)
        {
                putchar('2');
                button_state = CHOICE;
                debounced = 0;
                toggleLED(button4, 'C');
        }
        break;
//volume up and scroll up in playlist mode
case VOLUMEUP:
        if (pit_count%30 == 0) debounce(button2);
        if(mode==NORMAL)
        {
                if (push_flag)
                {
                        if(pit_count%100 == 0)
                        {
                                putchar('4');
//delay to prevent putchar from occuring mutliple times on the same tick
                                delay_ms(1);
                        }
                }
                else if(debounced)
                {
                        button_state = CHOICE;
                        debounced = 0;
                        toggleLED(button2, 'C');
                }
        }
        else if (mode==PLAYLIST)
        {
                //scroll playlist up
                if(debounced)
                {
                        //scroll playlist up
                        scroll=UP;
                        button_state = CHOICE;
                        debounced = 0;
                        toggleLED(button2, 'C');
                }
        }
        break;
//volume down and scroll down in playlist mode
case VOLUMEDOWN:
        if (pit_count%30 == 0) debounce(button1);
        if (mode==NORMAL)
        {
                if (push_flag)
                {
                        if(pit_count%100 == 0)
                        {
                                putchar('5');
//delay to prevent putchar from occuring mutliple times on the same tick
                                delay_ms(1);
                        }
                }
                else if(debounced)
                {
                        button_state = CHOICE;
                        debounced = 0;
                        toggleLED(button1, 'C');
                }
        }
        else if(mode==PLAYLIST)
        {
                //scroll playlist down
                if(debounced)
                {
                        //scroll playlist down
                        scroll = DOWN;
```

```
                        button_state = CHOICE;
                        debounced = 0;
                        toggleLED(button1, 'C');
                }
        }
        break;
//fast forward and skip to next song
case FFWD:
        if (pit_count%30 == 0) debounce(button3);
        if (push_flag)
        {
                if(pit_count%400 == 0)
                {
                        putchar('d');
                        repeat_flag = 1;
//delay to prevent putchar from occuring mutliple times on the same tick
                        delay_ms(1);
                }
        }
        else if(debounced)
        {
                if(repeat_flag == 0)
                        putchar('b');
                else repeat_flag = 0;
                button_state = CHOICE;
                debounced = 0;
                toggleLED(button3, 'C');
        }
        break;
//rewind and skip to prev song
case RWD:
        if (pit_count%30 == 0) debounce(button7);
        if (push_flag)
        {
                if(pit_count%400 == 0)
                {
                        putchar('e');
                        repeat_flag = 1;
//delay to prevent putchar from occuring mutliple times on the same tick
                        delay_ms(1);
                }
        }
        else if(debounced)
        {
                if(repeat_flag == 0)
                        putchar('c');
                else repeat_flag = 0;
                button_state = CHOICE;
                debounced = 0;
                toggleLED(button7, 'C');
        }
        break;
//change mode between playlist and normal mode
case TOGGLE_MODE:
        if (pit_count%30 == 0) debounce(button0);
        if (debounced)
        {
                if(mode==PLAYLIST)
                {
                        mode = NORMAL;
                        //set the start state of print to the play status
                        status = PLAY_STATUS;
                }
                else
                {
                        mode=PLAYLIST;
                        //set the start state of print to print the playlist
                        status = PLAYLIST_PRINT;
                }
                button_state = CHOICE;
                debounced = 0;
```

```
                                toggleLED(button0, 'C');
                        }
                        break;

                }
                //call check status every 50ms
                if(pit_count%50==0)
                        checkStatus();
        }
}


//this function is the LCD routines.
//it retrieves the needed info fromt the plugin
//and spits it out to the LCD one piece at a time
void checkStatus()
{
        unsigned char x;

        switch(status)
        {
                case PLAY_STATUS:
              //not waiting for any input
              if(wait_for_input == 0)
                        {
                        get_cmd_init();
                                putchar('7');  //send status request
                                wait_for_input = 1;
                        }
                        //data is ready to be read
                        if(cmd_ready==1)
                        {
                                sprintf(play_status_str,"%-s", format(cmd_str));
                                play_status_str[5]=0;
                                lcd_gotoxy(0,1);
                                lcd_puts(play_status_str);
                                wait_for_input = 0;
                                status = SONG_TITLE;
                                start_counter = 0;
                        }
                 break;
                 case SONG_TITLE:
                        //not waiting for any input
              if(wait_for_input == 0)
              {
                        get_cmd_init();
                                putchar('h');  //send songtitle request
                                wait_for_input = 1;
                        }
                        //data is ready to be read
                        if(cmd_ready==1)
                        {
                                strcpy(song_title_full,cmd_str); //copy the input data to the song title

//if same song is still playing and the whole or at least the end of the title didn't show last time
                                if(song_pos_timer==10)
                                {
                                        counter++;
                                        if(counter==2)
                                        {
                                if(strcmp(song_title_full, song_title_full_prev)==0
                                    && strpos(song_title_full_prev,'\0')>16
                                    && multi_space(song_title_print)==0)
                                                {
                                                        //change index so the song will seem to scroll
                                                        song_pos++;
                                                }
                                                //reset index
                                                else
                                                {
                                                        song_pos=0;
```

```
                                        song_pos_timer=0;
                                }
                                counter = 0;
                        }

                }
                else song_pos_timer++;
                //copy the first 16 characters of the song title.
                for(x=0;x<16;x++)
                {
                        song_title_print[x]=song_title_full[x+song_pos];
                }
                song_title_print[16]=0; //null terminate the string

                strcpy(song_title_full_prev,song_title_full);

                sprintf(lcd_buffer0,"%-s", format(song_title_print));
                lcd_buffer0[16]=0;

                lcd_gotoxy(0,0);
        lcd_puts(lcd_buffer0);
                wait_for_input = 0;
                status = TIME_STATUS;
                start_counter = 0;
        }
 break;
 case TIME_STATUS:
        if(wait_for_input == 0)
        {
        get_cmd_init();
                putchar('g');  //send songtitle request
                wait_for_input = 1;
        }
        //data is ready to be read
        if(cmd_ready==1)
        {
                sprintf(song_time, " %s",cmd_str);
                wait_for_input=0;
                status = BITRATE;
                start_counter = 0;
        }
        break;

 case BITRATE:
        if(wait_for_input == 0)
        {
        get_cmd_init();
                putchar('9');  //send bitrate request
                wait_for_input = 1;
        }
        //data is ready to be read
        if(cmd_ready==1)
        {
                sprintf(bit_rate, "%s",cmd_str);
                bit_rate[5]=0;
                sprintf(output, "%s  %s",song_time,bit_rate);
                lcd_gotoxy(5,1);
                lcd_puts(output);
                wait_for_input=0;
                status = PRINT_STATUS;
                start_counter = 0;
        }
        break;

 case PLAYLIST_PRINT:
        //if(pit_count%300==0){
        if(wait_for_input == 0)
{
        get_cmd_init();
        if(scroll==NONE)
        {
```

```
                                putchar('l');
                                }
                                else if(scroll==DOWN)
                                {
                                        putchar('j');
                                }
                                else if(scroll==UP)
                                {
                                        putchar('k');
                                }

                                scroll = NONE;
                                wait_for_input = 1;
                        }
                        //data is ready to be read
                        if(cmd_ready==1)
                        {
                                for(x=0;x<16;x++)
                                {
                                        pl_line0[x]=cmd_str[x];
                                        pl_line1[x]=cmd_str[x+16];
                                }

                                pl_line0[16]=0;
                                pl_line1[16]=0;

                                wait_for_input=0;
                                status = PRINT_PL_0;
                                start_counter = 0;
                        }
                        break;
                //print first song of playlist
                case PRINT_PL_0:
                        lcd_gotoxy(0,0);
                        lcd_puts(pl_line0);
                        status = PRINT_PL_1;
                        break;
                //print second song of playlist
                case PRINT_PL_1:
                        lcd_gotoxy(0,1);
                        lcd_puts(pl_line1);
                        status = PRINT_STATUS;
                        break;
                //resest to beginning state
                case PRINT_STATUS:
                        if(mode == PLAYLIST)
                                status = PLAYLIST_PRINT;
                        else  status = PLAY_STATUS;
                        break;
        }
}

//remove all null characters and replace with spaces
char * format(char string[])
{
        char x=0;
        char y;
        while(string[x]!='\0')
                x++;
        for(y=x;y<16;y++)
                string[y]=' ';
        string[16]=0;
        return string;
}

//count the num of consecutive empty spaces
//used for song scrolling
unsigned char multi_space(char string[])
{
        unsigned char x, count=0;
        for(x=13;x<16;x++)
```

```
        {
                if(string[x]==' ')
                        count ++;
        }
        if(count==3)
                return 1;
        else return 0;
}

//debounce button
//30ms ticks
void debounce(int button)
{
        switch(debounce_state)
        {
                case NO_PUSH:
                        if(~PINB == button) debounce_state=MAYBE_PUSH;
                        break;
                case MAYBE_PUSH:
                        if(~PINB == button)
                        {
                                debounce_state=PUSHED;
                                push_flag=1;
                        }
                        else debounce_state=NO_PUSH;
                        break;
                case PUSHED:
                        if(~PINB == button){}
                        else debounce_state=MAYBE_NO_PUSHED;
                        break;
                case MAYBE_NO_PUSHED:
                        if(~PINB == button) debounce_state=PUSHED;
                        else
                        {
                                push_flag=0;
                                debounce_state=NO_PUSH;
                                debounced = 1;        //flag set when completely debounced
                        }
                        break;
        }
}

//timer 0 interrupt service routine
interrupt [TIM0_COMP] void timer0_compare(void)
{
        //1ms ticks
        pit_count++;   //counter for checking debounce state machine

        //timeout counter
        if(start_counter==1)
                time_out_counter++;

        //if timeout
        if(time_out_counter==500)
        {
                //disable interrupt for now
                UCSRB.7 = 0;
                TIMSK=0;
                //re-init
                init();
        }



}

//function to toggle leds.
//easy to use, takes out some complexity in
//writing code
void toggleLED(unsigned int led, char port)
{
```

```
        if(port=='a' || port=='A')
                PORTA = PORTA^led;
        if(port=='b' || port=='B')
                PORTB = PORTB^led;
        if(port=='c' || port=='C')
                PORTC = PORTC^led;
}

void init(void)
{
        //set up the ports
        DDRD=0x00;      // PORT D is an input
        DDRB=0x00;     // PORT B is an input - buttons
        PORTB=0xff;    // PORT B off
        DDRC=0xff;     // PORT C is output - LEDs
        PORTC=0xff;  //port c is off
        DDRA=0xff;

         //USART
        UCSRB = 0x18;
        UBRRL = 103;

         //set up timer 0
        TIMSK=2;
        OCR0 = 250;    //set the compare re to 250 time ticks
        TCCR0=0b00001011;

        wait_for_input = 0;
        status = PLAY_STATUS;
        repeat_flag=0;
        mode = NORMAL;
        scroll = NONE;

        pit_count=0;
        counter = 0;
        song_pos_timer=0;

        start_counter = 0;
        time_out_counter = 0;
        debounced=0;        //button is debounced
        adjust_done_flag=0; //done making changes to freq

        debounce_state = NO_PUSH;
        button_state = CHOICE;

        lcd_init(LCDwidth);
        lcd_gotoxy(0,0);
        lcd_putsf("WinAmp Cont v1.0");
        lcd_gotoxy(0,1);
        lcd_putsf(" no connection");
        //crank up the ISRs
        #asm
                sei
        #endasm
}
```