ECE 2036 Lab #4 - Build an mbed MP3 Player

Due Dates: Wednesday, Oct 22 Section C
Friday, Oct 24 Section B

1. Overview

In this laboratory assignment, the mbed inventor's kit parts will be used to build a portable music player. There is an existing waveplayer hello world program that will play a wave file from the SD card using the speaker with the driver circuit connected to mbed's D/A output described in the waveplayer section at https://mbed.org/users/4180_1/notebook/using-a-speaker-for-audio-output/. Import the waveplayer hello world code example as your initial template. Watch the You Tube video (http://www.youtube.com/watch?v=VE-L6jybgkY&feature=player_embedded) on that web page to hear how it sounds. It would probably be a good idea to run the waveplayer hello world program first to verify that the SD card and speaker hardware is setup correctly. Don't forget that a "sample.wav" wave audio file is needed on the SD card. The speaker will not be quite as loud using the D/A output as the earlier lab that used the PWM output to drive the speaker.

In the wave_player class code used, a timer producing periodic interrupts (mbed Ticker API) is used to control the audio sample rate. These timer interrupts periodically activate a fast interrupt routine that outputs the next D/A value from a buffer and returns, while the wave player routine is reading and buffering audio sample values to be sent later to the D/A from a *.wav format file on the SD card. This audio data sample buffer uses C style dynamic memory allocation with *malloc* and *free*. This code is already included in the waveplayer hello world example. ***Your assignment will be to create and add a user interface that allows you to select which song to play on the LCD, play/stop, and adjust the volume using the pushbuttons.***

It might be a good idea and save time to use the same pin assignments and breakout board placement as the earlier mbed lab. In this lab you will need to add the micro SD card functionality. This is discussed in the next section. For modules that you have used in the skeleton code in previous labs you will need to copy over the appropriate header and implementation files into your working directory. For example, you can cut and paste the pushbutton and setup uLCD code from the earlier mbed lab and add it to the waveplayer helloworld example. Don't forget the uLCD and PinDetect *.h and *.cpp files need to be copied over into the project, and the include statements for them will need to be added in main.cpp along with the code to setup and activate the pushbutton callbacks. Copy and paste can be used for entire C++ source files in the right column in the mbed compiler window.

The player allows a user to select one of the files in the SD card's myMusic directory to play using pushbuttons and the uLCD. A sample code snippet is provided below that reads the filenames from this directory into a C++ vector of strings and prints all of the filenames out to

the PC terminal application window. There is an existing C++ structure setup, *dirent*, that can be used to read the directory entries that makes the code relatively short. This code assumes that the SD card file system is already setup in the program. This code shows how the filename strings can be read into a vector. It will need to be modified to display filenames individually on the uLCD rather than listing them on the PC.

```
 #include <vector>
……
vector<string> filenames; //filenames are stored in a vector string

void read_file_names(char *dir)
{
    DIR *dp;
    struct dirent *dirp;
    dp = opendir(dir);
//read all directory and file names in current directory into filename vector
    while((dirp = readdir(dp)) != NULL) {
        filenames.push_back(string(dirp->d_name));
    }
}
……put this code somewhere in main, it assumes SD filesystem is setup……
 // read file names into vector of strings
 read_file_names("/sd/myMusic");
 // print filename strings from vector using an iterator
 for(vector<string>::iterator it=filenames.begin(); it < filenames.end(); it++)
  {
    printf("%s\n\r",(*it).c_str());
  }
```

Several wave files will need to be placed on the SD card in the myMusic directory to provide test data. A minimum of four audio files are needed for the demo and you can select your own music or sound effect files. The Audacity audio editor is free and can be used to read in an audio MP3 or WAV file, convert it from stereo to mono, resample it to a 16Khz sample rate and export it as a 16-bit wave file. The *.wav file size is around 2MB for 1 minute of audio.

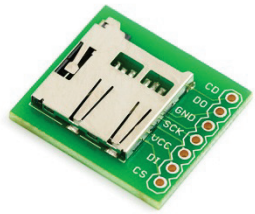
Some existing wave files may not play and will need to be converted using Audacity. Typically the problem is too high of a sample rate (>20Khz?) and they can be resampled at a lower rate in Audacity. The sample *.wav files can then be transferred to the micro SD card using a PC with the larger SD card reader adapter. Error messages are displayed with *printfs* to a PC terminal application window by the waveplayer code if you need to check them. Most wave file errors result in no speaker sound.

The bottleneck that limits the sample rate is the read time of the SD card. Serial transfers are used to the SD card. An mbed user in Japan has increased the SPI clock frequency in SDFileSystem.cpp from 1Mhz to 20Mhz while reading the SD card file in the *SDFileSystem* class and claims even higher sample rates, but it will work ‘as-is’ to at least 16Khz. Increasing the clock might only work on some SD cards and the breadboard jumper wires would need to be short. Mbed’s small internal flash drive is too slow for the wave player and the external SD card must be used instead.

The existing waveplayer hello world example code in main.cpp plays using a string constant (i.e. “sample.wav”) argument for the filename, this code will need a minor modification to accept a C

style string variable as the argument for the filename to play. Note that C++ strings are stored a bit differently than C strings (no null character at end), but there is a function to convert them, *c_str()*, that can be found in the earlier filename example code. Don't forget that the filename to play will need a prefix with the full path name ("/sd/myMusic/"). Also the file to play needs to be opened each time before playing and once it stops playing, it must be closed.

2. Using the Micro SD Card



The micro SD card (http://mbed.org/cookbook/SD-Card-File-System) or a USB flash drive can be used for storage. For this lab you will use the SD card. Please use the constructor call in the below sample code that WRITES a simple message to a file on your SD card to determine the pin connections that you need. The micro SD card comes with a larger SD card adapter and it should be inside the adapter in your kit. Most laptop PCs have an SD card reader slot, if not the TA has some USB adapters in the lab. The micro SD can be put into the larger SD card adapter to read or write the data files on a PC. An example file that you might use to verify that your SD card is working is given below. It is taken from the mBED cookbook website SD card file system code example.

```
#include "mbed.h"
#include "SDFileSystem.h"
SDFileSystem sd(p5, p6, p7, p8, "sd"); // the pinout on the
mbed Cool Components workshop board

int main() {
    printf("Hello World!\n");

    mkdir("/sd/mydir", 0777);

    FILE *fp = fopen("/sd/mydir/sdtest.txt", "w");
    if(fp == NULL) {
        error("Could not open file for write\n");
    }
    fprintf(fp, "Hello fun SD Card World!");
    fclose(fp);

    printf("Goodbye World!\n");
}
```

3. Laboratory Requirements

Basic WavePlayer (75%)

Use two pushbuttons to scroll forwards or backwards through the filenames string vector and display the current filename on line 2 of the text LCD (you can assume filenames are less than the 16 characters on one line in the LCD). Do not display the filename's ".wav" file extension on the LCD. A handy function to remove the ".wav" file extension is *substr*(). When advancing beyond the last filename in the vector, it should wrap around back to the beginning. A third pushbutton will be used to start playing the filename currently displayed on the text LCD. Right before the song starts playing, line 4 on the LCD displays the message "Song Playing". If the play pushbutton is hit again while a song is playing it stops the player, clears line 4 on the LCD and goes back to the select a song to play state in the text LCD. When a song finishes playing, it also returns to the select a song state. It probably makes sense to setup all of the pushbuttons using a callback as was done in the earlier lab. Note: When the SD card files are created on MACs, you will want to add code to ignore the extra hidden files that start with ".".

Since the wave_player class C++ code is setup as a separately compiled library and linked after compilation to the main program, it is not possible to share global variables (set by pushbuttons) with main.cpp. The play/stop function variable and the volume variable for the last part will need to be passed to the wave_player class as a new argument using a pointer or reference. The play method in the wave_player class will need additional arguments added to pass these. Since the play/stop value changes with pushbutton interrupts asynchronously, a pointer is needed and pass by value will not work to stop a song while it is playing. The *volatile* keyword is not critical but is still a good idea on these two variables (play/stop and volume) since the main program always writes them and the wave player class only reads them. The *for (slice=0…* loop is a good place to break out of the player code for the stop pushbutton as it turns off interrupts and the D/A when exiting this loop.

Add volume control (10%)

Use the fourth pushbutton to add volume control. For faster operation, the wave_player dac_out method actually sends out an unsigned integer value (i.e., AnalogOut write_u16 method) to the D/A instead of using scaled floating point numbers (this code is found near the end of wave_player.cpp). Find this line of code and add a scaling operation that multiples the D/A value by (16 - volume) and then divides the D/A value by 16 (with a shift of 4-bits right or ">>4") as the value is sent to the D/A to adjust the volume. Note: integer operations are faster than floating point and shifts are faster than a divide. The fourth volume control pushbutton cycles through the values (0..15) using mod (i.e., "% 16") and sets the volume variable to control the volume. The initial default volume setting should be full volume (i.e., no scaling down). Add an argument to pass a pointer or reference to the volume variable to the play method in wave_player.cpp. The play method can copy the pointer to a global variable (in waveplayer.cpp) so that the interrupt routine, dac_out, will then be able to access it for scaling. Since the volume value changes with pushbutton interrupts asynchronously, a pointer is needed (i.e., copy by value will not work).

Upgrade to a Boom Box (5%)

It is possible to connect to amplified PC speakers (or use a headphone with a driver or audio amplifier circuit) using the audio jack breakout board in your parts kit. See the "Using PC speakers with mbed" section at https://mbed.org/users/4180_1/notebook/using-a-speaker-for-audio-output/. The lab TAs can loan you the extra resistors and capacitor needed. There is a set of PC speakers available in the Klaus ECE PC lab for checkoff demos.

Add SD card detect feature (5%)

As you may have noticed, trying to read or write files without an SD card plugged in can result in the mbed flashing "blue LEDs of death". It is possible to read a pin on the SD breakout board that detects when an SD card is inserted. Details on how this pin functions can be found near the end of the SD card file system cookbook page. Modify the *SDFileSystem* class code to add another argument for the new CD (Card Detect) pin (currently unused in code and not hooked up in hardware). Use this new pin argument in the class to set up the pin and add a new member function, *SD_inserted*, that returns true if an SD card is inserted and false if not. Read the handbook page on DigitalIn to see how to configure the internal "pullup" resistor using the *mode* method. Modify your main program code so that this new class member function is used when the program first starts. If there is no SD card present, print "Insert SD Card" on line 2 of the Text LCD and loop constantly checking for the SD card to be inserted. Once it is inserted, clear the message and start the player program that selects a song using the pushbuttons. You can assume that the SD card is not removed once the program passes this initial check. A short time delay will be required after the SD card is inserted to allow it to power up correctly after insertion (there is actually a tiny microprocessor with firmware inside the SD card) and a call to *disk_initialize()* also helps to make it operate more reliably after a power on insertion.

Add Multiple File names to Menu Feature (5%)

Display more than one song file name option at a time on the uLCD screen (at least the four required files, and you can assume only 1 screen full of song files are on the SD card, if you want to). The pushbutton should change both the text and text background color of the selected song in the list on the LCD. Clear out the file name display on the LCD when displaying the "Song Playing" message. The list should reappear whenever a song stops playing.

Extra Credit: uLCD Artistic Merit (+5% images or +10% videos - pick one option only)

Instead of just displaying the dull "Song Playing" message on the uLCD, display a unique 128 by 128 color image (5%) or video clip (10%) for each song. The uLCD wiki page has information on the additional image tools and steps required. For images and videos, it should be something relevant to the song. Perhaps a picture of the artist or a music video, if one is available. This will require using the SD card in the uLCD for the image or video files, so a second SD card is needed or you could even hookup a USB flash drive (using your parts kit USB breakout board) to hold the audio files. Stopping a video while it is playing may require an LCD reset. Another possible solution for stop might be the video display frame command used in a loop.

Extra Credit Hints: Putting Images and Videos on the SD card

To use the SD card unformatted partition tool and the graphics conversion tool, download, install, and start the 4D Workshop IDE. Select **File->new** and the scroll down the LCD pictures to **select the uLCD-144**. Click next and then select serial -> **next**. The tools menu now appears on the top line of the IDE and this is where the RMPet tool is found. You can use it to create a RAW unformatted SD card or change the partition setup (they come formatted and can easily be reformatted again – see mbed cookbook SD filesystem's format link to reformat an SD card to factory default settings).

Next, images and videos can be converted and written to a RAW unformatted SD card using the Graphics Composer tool (also under tools menu). Select 128 by 128 screen size (lower right) and just add and edit each image or video needed in the graphics composer tool. Unclick Ignore screen size constraints to downsize each image or video for the 128 by 128 LCD. When all videos and images have been added and resized using edit, click the IC icon to build the file and it will write it out to your (unformatted) SD card. **Note:** Conversion of long high-res videos requires a fast PC with a lot of memory. It might be a good idea to cut down on the resolution and length of videos with a video editor prior to using the LCD video conversion tool.

The SD card can then be plugged into the socket on the back of the uLCD. The tricky part now is keeping track of the sector address on the SD card where each image or video is stored (unformatted SD cards do not have a directory, you have to keep track of this sector address data manually and put it in the C++ code to display the image or video). Several of the IDE's project files list the starting sector address of each image. Click on the "GC" icon to see a list of the images and sector addresses.

Extra Credit Hints: Displaying Videos and Playing audio at the same time

The functions to display a video and play audio do not return until the video or audio playback is complete, so they cannot be both started and running at the same time. To fix this, the display_video command code in the uLCD driver could be changed not to wait for the final command acknowledge (ACK) character to be sent back from the LCD (it is only sent back by LCD when video playback completes). This would take a different version of the writeCOMMAND function that it calls - the other commands still need to wait, so you would need a new writeCOMMAND_NOACK only used by display video. – This version would not wait to read back the final ack character from the LCD. Code when the player and video stops would still need to read in the final ACK character later with a getc to avoid messing up any LCD commands that follow.

See ::display_video in https://mbed.org/users/4180_1/code/4DGL-uLCD-SE/file/e39a44de229a/uLCD_4DGL_Media.cpp to find the display_video code. WriteCOMMAND is in another include file https://mbed.org/users/4180_1/code/4DGL-uLCD-SE/file/e39a44de229a/uLCD_4DGL_main.cpp. You can change this code in your local copy of the code

ECE 2036 Lab 4 Check-Off Sheet

Student Name: _____

Section/Professor: _____

Demonstration	**TA Signoff**
Basic Waveplayer (75%)	
Add Volume Control (10%)	
Upgrade to Boom Box (5%)	
Add SD Card Detect (5%)	
Multiple Filename Menu Feature (5%)	
uLCD Artistic Merit (+5 or +10%)	
Late Penalty (-20% per day)	
Demo Late Penalty (-10 % per day)	

Lab Submission Instructions:

Each lab must be demonstrated to the TA and submitted on T-Square on or before the due date and time. Students should be finished with the lab before the start of TA office hours on the due date, so that they may get the lab checked-off on the due date and then submit the final code on T-Square. Of course, students may get checked-off and submit their code early! Students cannot expect to get TA help on lab due dates, as their time will be dedicated to lab check-offs. Only one formal check-off attempt is allowed per student, and the entire lab must be checked-off at once.

If you cannot demo the lab in office hours on or before the due date and time, you must come demo during one of the next five TA office hour days after submitting your code online. There will be a 20% per day penalty if the lab is not submitted on T-Square on time. There will be an additional 10% per day per day penalty if you do not demonstrate the submitted code to the TA within five TA office hour days of the submission deadline.