

Audion

Project Report

Grace Tian, Sarah Kang, Aishwarya Srinivas

Group 9

ECE 150

Table of Contents

Project Overview...…………………………………………………………........…3

System Design……………………………………………………………....….......4

Software Design………………....……...………………………………………….9

Testing………………………………………..………………………………..….13

Limitations………………………………………….…………………………..…16

Reflection…….……………………………………………………………………16

Appendix……………………………………………………………………...…..17

OVERVIEW

PROJECT DESCRIPTION

The main idea of this project is to create a music player that stores WAV music files in an USB drive to be played through the Omega 2 into a speaker. The user can interact with this system through the use of the pushbuttons supported in our hardware. This embedded system is set up through C++ code. The program read all the files from a specific directory onto an array. A specific file or song can be called from the array according to the user’s needs.

WHAT IF IT COULD DO EVERYTHING?

Ideally, this system would be able to support many more codecs, such as MP3 (one of the most common codecs), as well as lossless codecs such as FLAC. The use of the OLED expansion would also be vital to provide a visual interface for what song is playing, as well as to display any sorting tags such as artist, album, and genre.

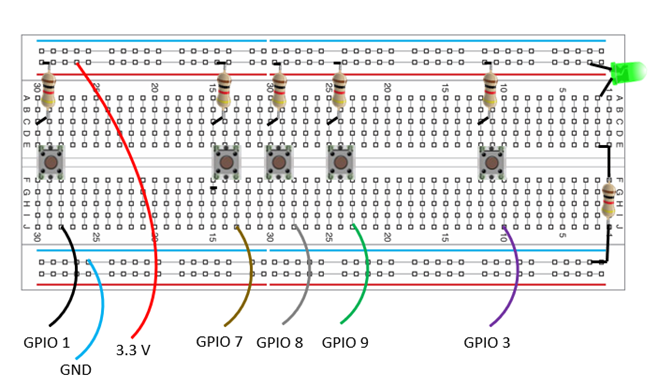
SUBSET

We built a music player that attaches to a bluetooth speaker. Music is stored in a USB key, which in turn, is attached to the Onion Power Dock through the USB port. Jumper wires connect the Onion dock to components on the breadboard. The breadboard comprises of two LEDs and five 3 mm push buttons, as well as their corresponding resistors. The user can press the power button to turn the music player off. A green LED light will be lit as long as the system is running. To play the song, or skip to the next or previous song, the user can click on their corresponding buttons present on the board. The user would also be able to interact with this system through the interface present on the command line terminal. Some of these options include listing the current playlist of songs, listing the songs that were favorited, choosing a random song (shuffle) and quitting the system. The user could also choose ‘shuffle’ to be played as well.

SYSTEM DESIGN

HARDWARE

Most of our hardware components are present on the breadboard. This includes 3mm push buttons, LEDs, and their corresponding resistors, as well as jumper wires to connect them to the pins on the Omega dock. Jumper wires connect the GPIO and pushbuttons and a 10k resistor diagonally connects the pushbutton to the 3.3 V. Below is a diagram of the system setup:

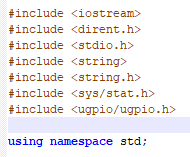


A USB hub and the speaker is connected to the Omega. A USB drive containing the songs are connected to the USB hub.

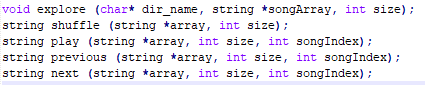
SOFTWARE

The main outline of the code is as follows:

LIBRARIES



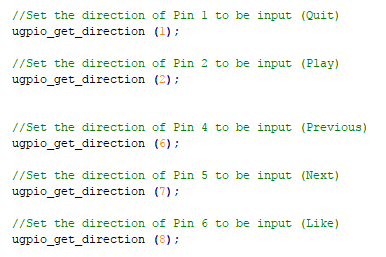
FUNCTIONS



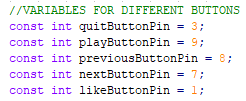
* To read the files from the designated directory onto an array
* To return a random song from the array
* To return a song from the playlist to be played
* To reduce the song index and return the previous song from the playlist
* To  raise the song index and return the next song

MAIN

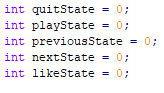
* Setting the direction of the GPIO pins.



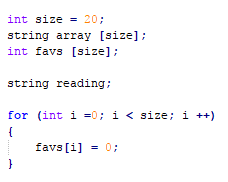
* Declaring variables for the buttons associated with their corresponding GPIO pin numbers.



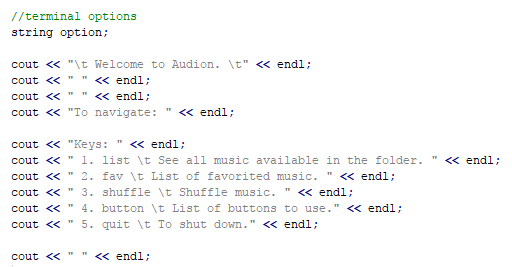
* Declaring the states at which the system is at. I.e. which buttons were pressed at that particular time.



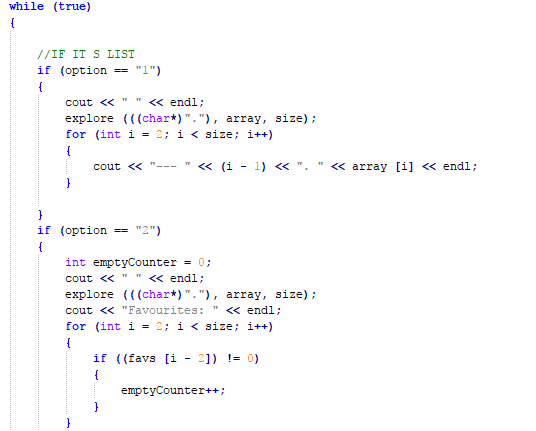
* Declaring the playlist array and initializing the favs array to be 0.

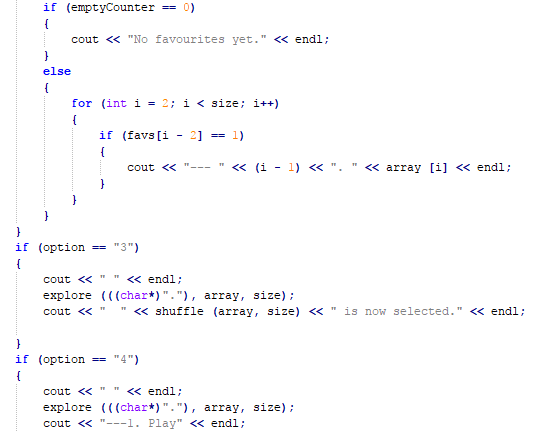


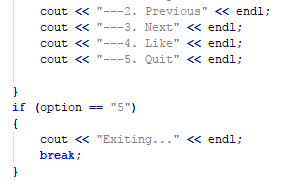
* Text to be displayed on the screen. I.e. the list of commands the user can enter.



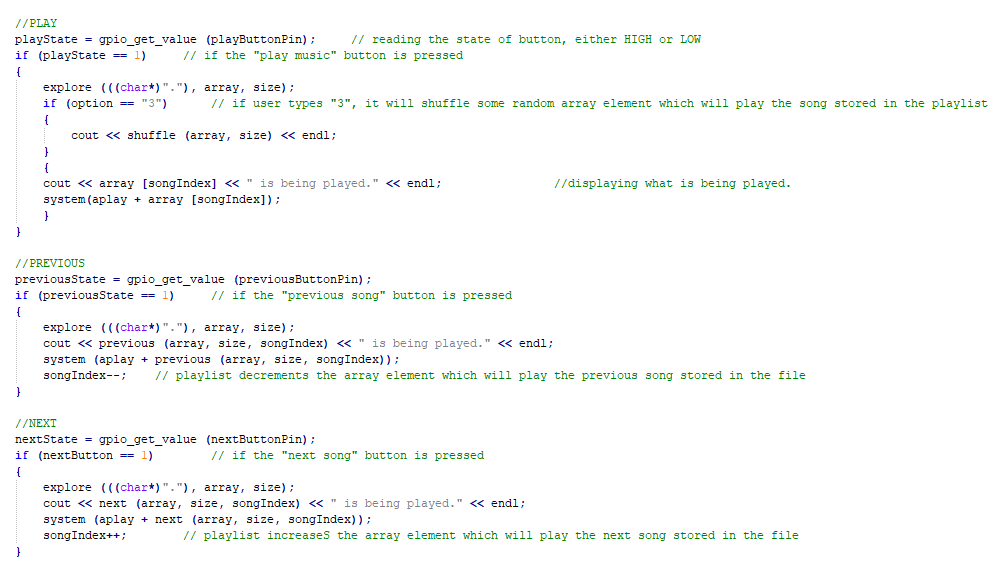
* If user types one of these options, an action would occur in a loop.

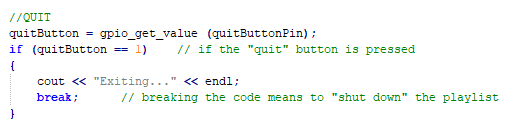






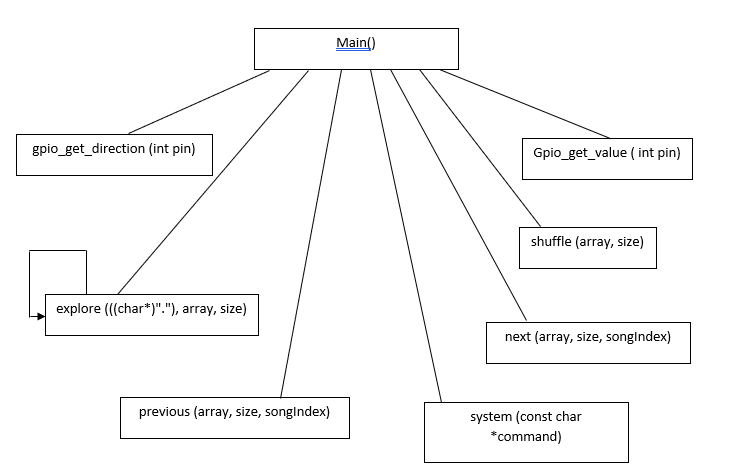
* A loop for the actual playing of the music and for when the buttons on the breadboard, are pressed.





SOFTWARE DESIGN

FUNCTION CALL TREE



STRUCTS

DIRENT -  This is a part of the dirent.h library. Its members are:

ino\_t  d\_ino          (file serial number)  
char   d\_name[]    (name of entry)

The serial number would later relate to the song index number of the file in the playlist array.

STAT INFO - This struct gets the properties about a file such as id, name, size, type, etc. This struct is a part of the sys/stat.h standard library.

INTERFACE DEFINITION

Below is a screenshot of how the user interface would look in the Onion terminal:

Options are listed and the user can enter a number corresponding to the options given: list, fav, shuffle and quit.

LIST - This will show all of the files that are read from the directory and put into the array.

FAV -  If the user presses on the “like” button when a song is playing, favs [songIndex] becomes true (“1”). Each song has its own index number. The user can type in ‘fav’ to see the list of songs that were liked.

SHUFFLE - This will return a random song from the playlist to be played.

QUIT - To quit the system and get out of the loop, the user can type ‘quit’.

LIBRARIES

* #include <stdlib.h>
* #include <iostream>
* #include <dirent.h>
* #include <time.h>
* #include <ugpio/ugpio.h> - external
* #include <alsa/asoundlib.h> - external
* #include <sys/types.h>

DATA ELEMENTS

**songArray (playlist)**

This is the main array that would act as a playlist storing all the files that are in .wav format. This string array only stores the names of the files.

**randNum**

A random index number from 0 to (size of the playlist - 1), is returned. Then, the element in the playlist with the same index as the random number is returned. This is how a shuffle works.

**Size**

There will be a certain amount of memory or space allocated for the songs. This will be the size of the songArray. This size is equal to 50 at most.

**songIndex**

In the main loop in the program, this integer value acts as a counter, keeping track of which song is being played. It will be incremented if the file is played, or if the user clicks on the next button. It is decremented if the previous button is pressed.

**ugpio\_get\_direction**

This sets the direction of the pins connected to the OMEGA. The pins connected to the LEDs are output, while the buttons are input.

**Buttons**

There are a total of 5 buttons that are to be used in this system: play, previous, next, like and quit. These functions are typically present on a MP3 player.

**Favs [ ]**

The favs array is initially set to 0 at first. If the user clicks on the like button, favs [songIndex] = 1. The songIndex also refers to the song in the playlist array with the same index. This particular song is favorited.

METHODS

**explore(char\* dir\_name, string \*songArray, int size)**

This is a function that reads through the specified number of files in a folder or directory and stores the file names into an array that would act as a playlist. This function implements the dirent.h library. As there would be a limit for the number of songs to be in the playlist, that many entries are read from the folder using readdir ().

**next()**

This function increases the index in the song array list. This function is called if the user presses the button for next song.

**previous()**

This function decreases the index in the song array list to go to the previous song. The function is called if the button for previous song has been pressed.

**shuffle()**

A random song will be chosen from the playlist and will be played. The rand() function chooses a value between 0 and the the array size.

**quit()**

To stop all function and also to break out of the loop, quit is called if the button for quit is pressed. This will stop the song from playing and shut down the terminal.

**play()**

This function returns the song with the current song index to be played.

**Ugoio\_get\_value**

This function is a part of the ugpio library and it gets the input value from a pin. This is mainly used with the buttons. If the button is pressed, it reads in 1.

**ugpio\_get\_direction()**

This function is for setting up the input or output modes of the pins attached to the Omega. The pins attached to the LEDS are output, while the ones attached to the buttons are input.

SYSTEM-INDEPENDENT COMPONENTS

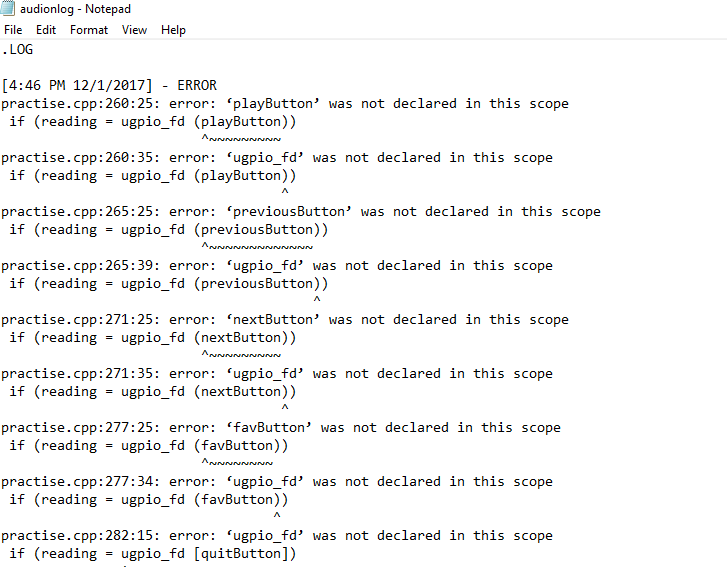
The functions and variables associated with files from the directory. These include the explore, play, previous, shuffle and next function. The options listed on the terminal and the ability for the user to navigate through these options is all system independent.

SYSTEM-DEPENDENT COMPONENTS

The variables and functions involved with the GPIO pins are dependent. These include the functions that use the ugpio library and storing the values from the pins into the state variables.

LOGGING

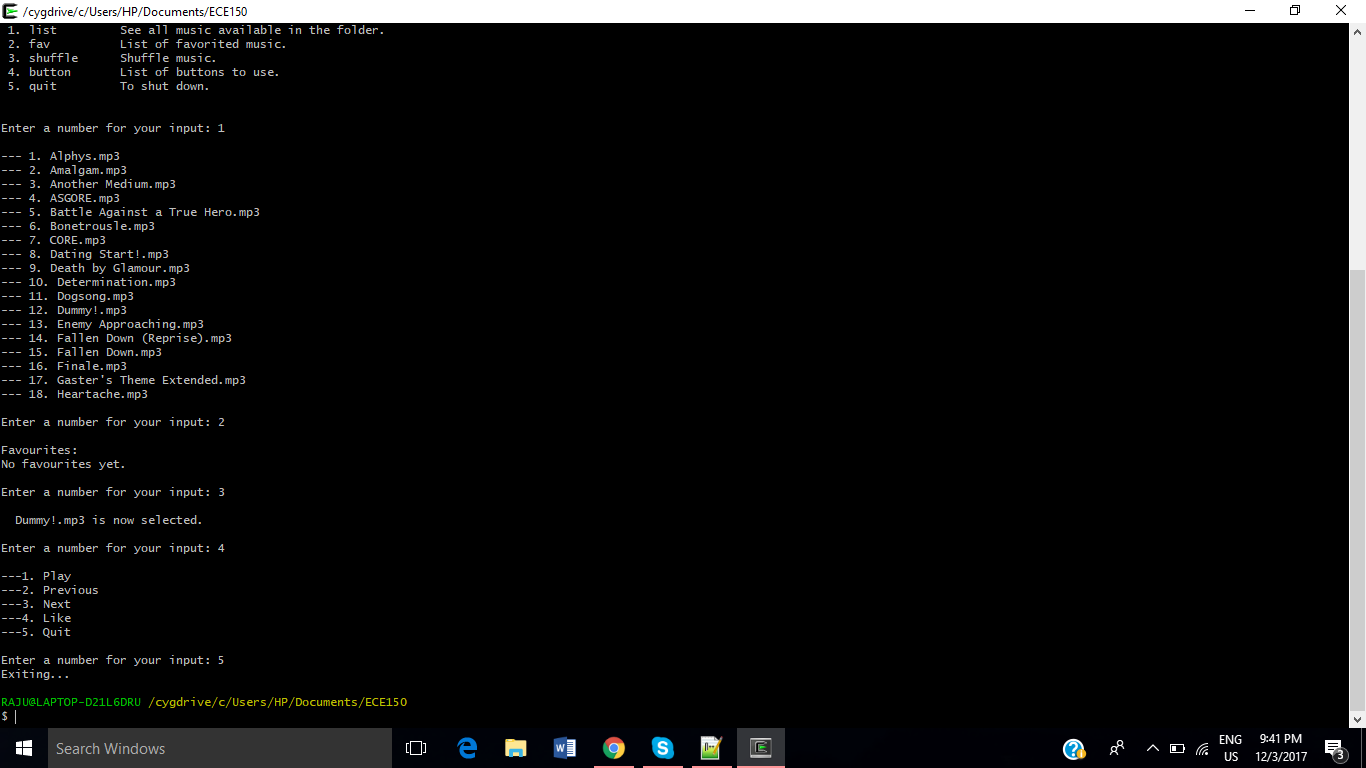
Logs were typed manually as follows:



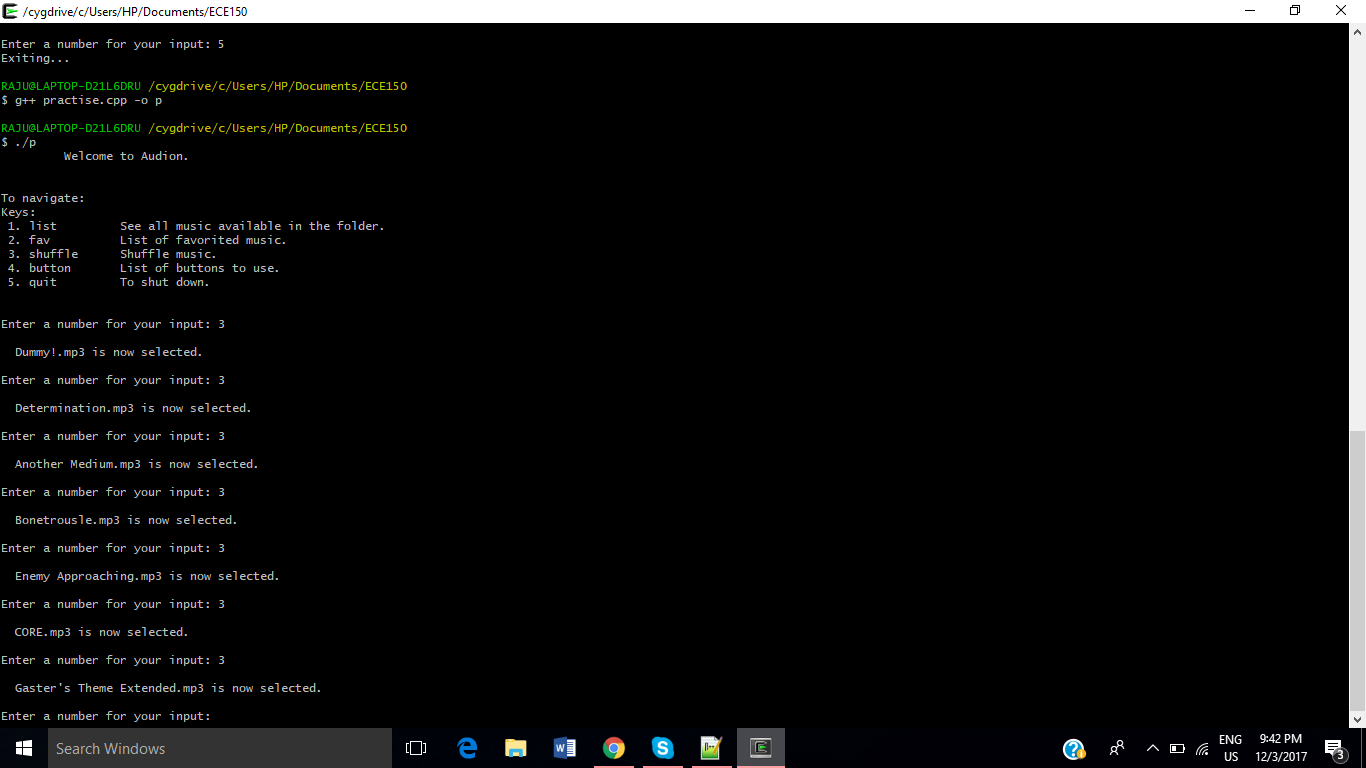
SOURCE CODE

The source code is Audion.cpp.

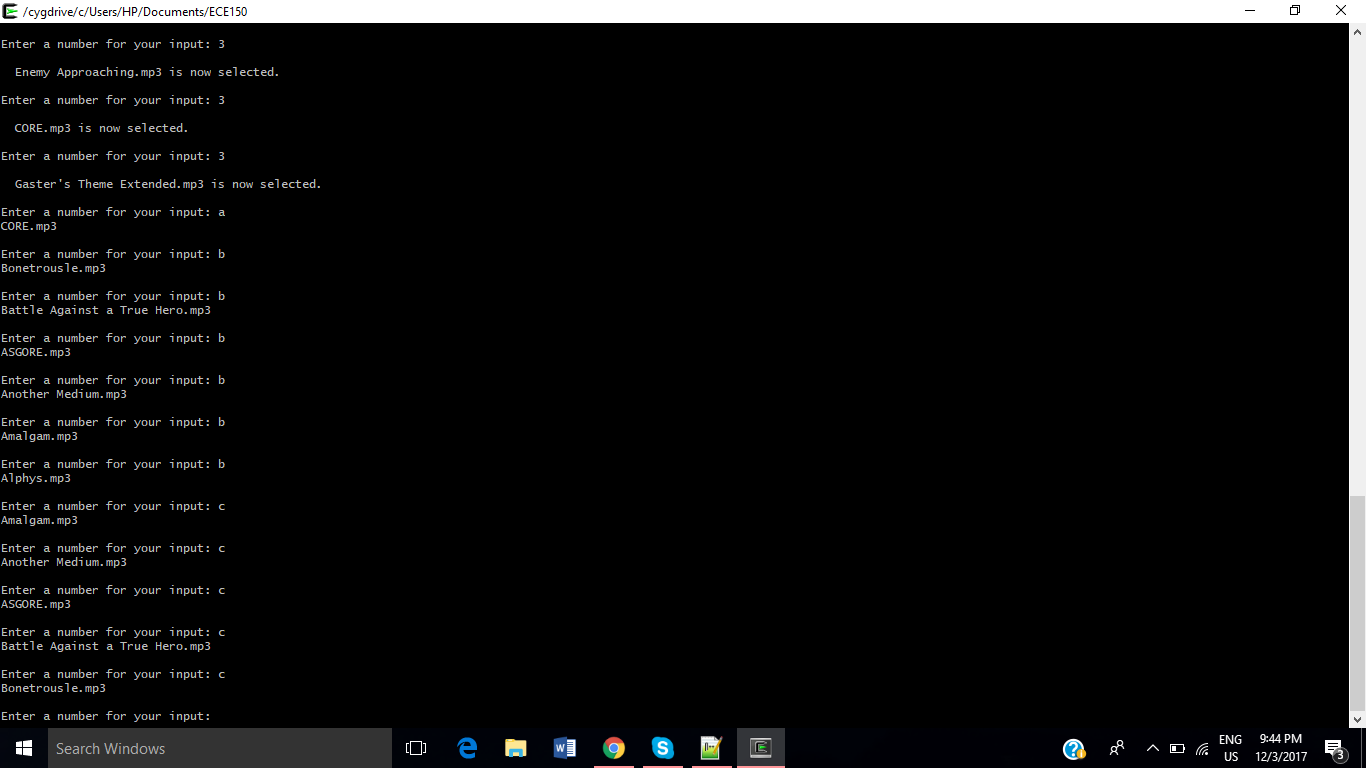
TESTING



* The first part of the program, that is to use the commands provided by the user, is tested. These commands are numbers from 1 to 5, corresponding to list, fav, shuffle, button and quit. In the above screenshot, numbers are entered consecutively. For ‘list’, to make sure that 18 items are listed, a random folder is set as the directory to be read. Seeing the file names on the terminal show that the explore function works properly.



* To make sure that the same command could be entered within the loop without any error, the shuffle command is re-entered a couple of times. This shows that the loop functions properly (until quit is entered).



* The second part of the program is to make sure that the play, previous and next methods work properly. Testing was done in the Cygwin compiler, so the ugpio library was not used in this version of the program. The input from the pins would have to be read in this part of the loop, but instead, string inputs are used, for testing. Here, ‘b’ represent previous and this function works as the song index decrements by 1 each time ‘b’ is entered. The same happen for the next function by the index increments by 1.

LIMITATIONS

* This system can only play .wav files as the alsa library only supports these kinds of files.
* The explore function () can read every file from the folder. So, it is assumed that all the files in the directory to be read are music files.
* It is not possible to pause songs as the alsa library does not have this function.
* There will be a limit on the memory space that will be allocated to the files. For simplicity, the folder can only contain 50 files.
* The user interface is confined to the Onion terminal and because of time constraints, the OLED expansion could not be displayed.
* The Onion Omega only supports digital output and input, so sensors could not be used.
* The program compiles perfectly on the Cygwin and Linux compilers.

REFLECTION

More research should have been done on the project we chose before submitting it for the proposal. We did not realize that it does not support audio codes other than wave due to Omega2 hardware limitations. Alsa doesn’t have the function to pause the music playlist and we were not able to implement a delay function. More research should have been done on the capabilities of the Omega board. We felt like we could have use other libraries. We wish we could have begun researching earlier so we would not be in a rush. We might have gotten more information and maybe have better solutions to some of our current problems. We are glad that we were able to apply many ECE 150 concepts into the code writing of the project (eg. struct, file, array, etc). Despite some functions not working on the omega, it is great to see that they were able to run on Linux.

APPENDIX

SOURCE CODE

Audion.cpp

The code is present in the same Dropbox folder as this report as “Audion.cpp”.

PEER CONTRIBUTION

SARAH KANG -

* Write code to play music using the alsa library.
* Contributed to limitations and project overview of report.
* Assembled audio hardware component of project.
* Researched and installed audio related libraries.
* Debugging and logging.

YUNFEI TIAN -

* Set up the hardware circuits (breadboard, LEDs, pushbutton buttons and pins).
* Drew the hardware diagram.
* Wrote code related to GPIO pins and reading its values.
* Wrote the hardware portion on the final report .

AISHWARYA SRINIVAS -

* Write outline of code and the interface on the terminal.
* Write all functions in code.
* Parts of report completed: Software Design, Limitations, Testing, Function Call tree.
* Logging.