

Brass2Go

DOCUMENTATION + USER MANUAL



Authors:

Usman Zia

Luke Garland

Toshiro Taperek

Developed for:

ILS – ENEL 300

Schulich School of Engineering

April 17, 2019

Table of Contents

1. Introduction	3
1.1. Project Introduction	
1.2. Key Stakeholders	
1.3. Target Market	
1.4. Persona Description	
1.5. Concept Description	
1.6. Fundamental Project Requirements	
2. Product Constraints	8
2.1. Design Limitations	
2.2. Technical Limitations	
2.3. Financial Budget	
3. Product Design	10
3.1. Circuit Design	
3.2. Hardware/Software Prerequisites	
3.3. PIC Programming	
4. Product Development Cycles	18
5. Project Management	19
6. Validation and Testing	22
6.1. Hardware Testing	
6.2. Software Testing	
7. Appendices	23
7.1. References	
7.2. User Manual	

1. Introduction

1.1. Project Introduction

Brass2Go is the result of the final project outlined in ENEL 300, a 2-year Electrical Engineering Course offered by the Schulich School of Engineering. This final project will aid in building a practical foundation in the listed courses taken in the Winter Semester of 2019:

- ENEL 343 – Circuits II
- ENEL 327 – Signals and Transforms
- ENEL 361 – Electronics
- ENEL 369 – Computer Organization

This final project has been assigned on the day of Jan 07th, 2019 and demonstration deadline is on the day of April 18th, 2019. The project requirements have been set by instructors for the above courses and Dr. Laleh Behjat.

1.2. Key Stakeholders

The key stakeholders of this final project have been listed below:

- Schulich School of Engineering
- Integrated Learning Stream – Pilot Program
- Faculty of Electrical and Computer Engineering
- Instructors of the courses listed in section 1.1.
- Industry Experts and Judges

1.3. Target Market

The Target Market for “Brass2Go” will not be limited to professional brass instrument musicians but also for hobbyist musicians. The music industry has been significantly growing within the past few years peaking at 5.9% in 2016, the highest to date, with revenues capping at \$15.7 Billion (USD) [4]. Our product “Brass2Go” is a highly innovative learning tool with error detection algorithms making it a new entry into the market. Given the competitive industry, our product will have an easier market takeover as this will be the first of its kind.

1.4. Persona Description

The “Brass2Go” has been created for the 90% of talented musicians that are not even known by the industry and are working part time jobs just to keep funding their music careers. Thus, we considered the affordability of the product and created “Brass2Go” with low cost parts.

The following is a common persona for our product:

A talented university student that dreams of joining a professional trumpet band and showcase her art of music. However, to accomplish her dream she juggles commitments between her education and practicing her trumpet. She believes that hard work and determination is the key to success and desires to find innovative methods of practicing her instruments while commuting back and forth to school.

The following is how the common persona intends on using:

She is on her way to school that consists of an hour-long C-train commute. As an opportunistic person she desires to practice her trumpet which is not possible due to the large size and respect for others on the train. She decides on investing in a portable music player but is unable to improve her skills as no insights are available. Brass2Go provides her with the opportunity to practice on the train every day to school resulting in 240 more hours of practice per academic semester. With our music software connected with Brass2Go she is now able to spot her mistakes and make improvements and is on her way to a rising trumpet prodigy!

The following is the description from one of our valued users – Salah Sheikh:

“This idea is great as I am always looking for learning tools to improve my muscle memory. Also, I was shocked that it could also tell me how many mistakes I made and my accuracy rating! This I believe would definitely help and I can keep doing it while on commute!”

USER FEEDBACK CATALOGUE:

Luke Garland; USMAN ZIA; Toshi Taperek ✍

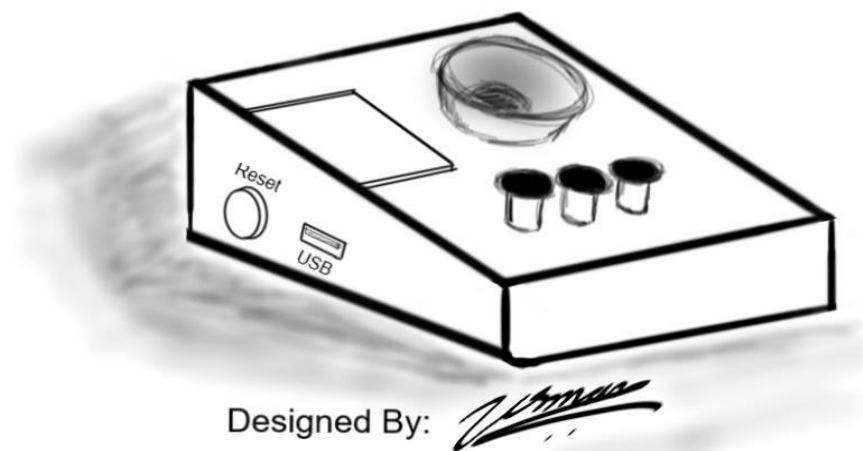
change your timer function to produce the output so its easy to stop and start the timer and check when delays happened.
Good idea, make sure to talk to Jason maybe about the code to play everything at the same and recording!
If the bits are too data intensive why don't you just use less data, it might sound worse but then you can handle i/o easier.
The internal timer on the PIC controller is somewhat inaccurate. You might consider adding an external crystal to better track delays.
Figure out the timing by talking to Paul about it
16bit audio sounds impressive!
What is the justification for using 16 bit audio?
There may be an external timing piece for the PIC that is available to help with the timing accuracy of the button presses.
Maybe chat with group 4 to see how they are dealing with their data intensive challenge (Cameron etc.), or group 3 (walking stick) to see if they are having timing issues.
Have you tried outsourcing your processing components to another device like an Arduino or a Raspberry Pi? It would make your job significantly easier.
Talk to Jason about having enough resources on the PIC to be able to do so many things at once.
I would recommend building the project with 8-bit audio first as the PIC might not have enough RAM to complete all of the tasks you are trying to complete.
You might be able to timestamp via interrupt, takes a load off the processor since it is computed externally. Lowering your sample rate can allow more to be completed if needed
Maybe try playing around with clock speed. Maybe have a separate clock for timing your checks on buttons and playing the sound. Or maybe a timer for timeout.
You have the exact same problem as us so we can provide our sympathies =)
If the issue is the PIC not being able to output and read inputs quickly enough, maybe scaling back to 8 bit audio can give you a bit more headroom. It won't sound as good, but given the speaker we're using, it might not ultimately matter.

1.5. Concept Description

Brass2Go is a portable learning tool which allows a musician to practice the rhythm of a song on the go and hence excel at his performance by data driven insights. Unlike other instruments that provide no insights for improvements in a song and are not portable.

The idea behind Brass2Go is for the 90% of the unknown musicians that work jobs and need more time to practice their skills and compete with the 1% of successful musicians with significantly higher access to resources. Brass2Go will aid in providing the necessary instrument functionalities with error detection algorithms for feedback resulting in major improvements for its users.

The inspiration behind the concept is for music fanatics to have access to a broader range of musicians thus adding creativity to the music industry currently dominated by a few selected individuals.



1.6. Fundamental Project Requirements

The “Brass2Go” has been developed under the strict requirements of ILS 2019 project guidelines:

- The financial budget allocated for the project should not under any circumstances exceed \$60 (CAD).
- The project must be based on the PIC16F1778 Microprocessor.
- The end product must have the ability to play audio and require some sort of user interaction.

2. Product Constraints

2.1. Design Limitations

The following are self-defined design constraints in order to provide basic functionalities. These design constraints were imperative during the development of Brass2Go as the central idea always had to take these limitations into consideration.

Table 1: Self-defined design constraints of Brass2Go

	Design Constraints – Description
1	Programming onto the PIC16F1778 Microcontroller
2	Speaker and Head-phone output provided
3	Must be able to output audio files accurately

2.2. Technical Limitations

The Technical Constraints for this final project was outlined in the ENEL 300 – Project Rubric and the following table summarizes key constraints that directly affects the development of Brass2Go.

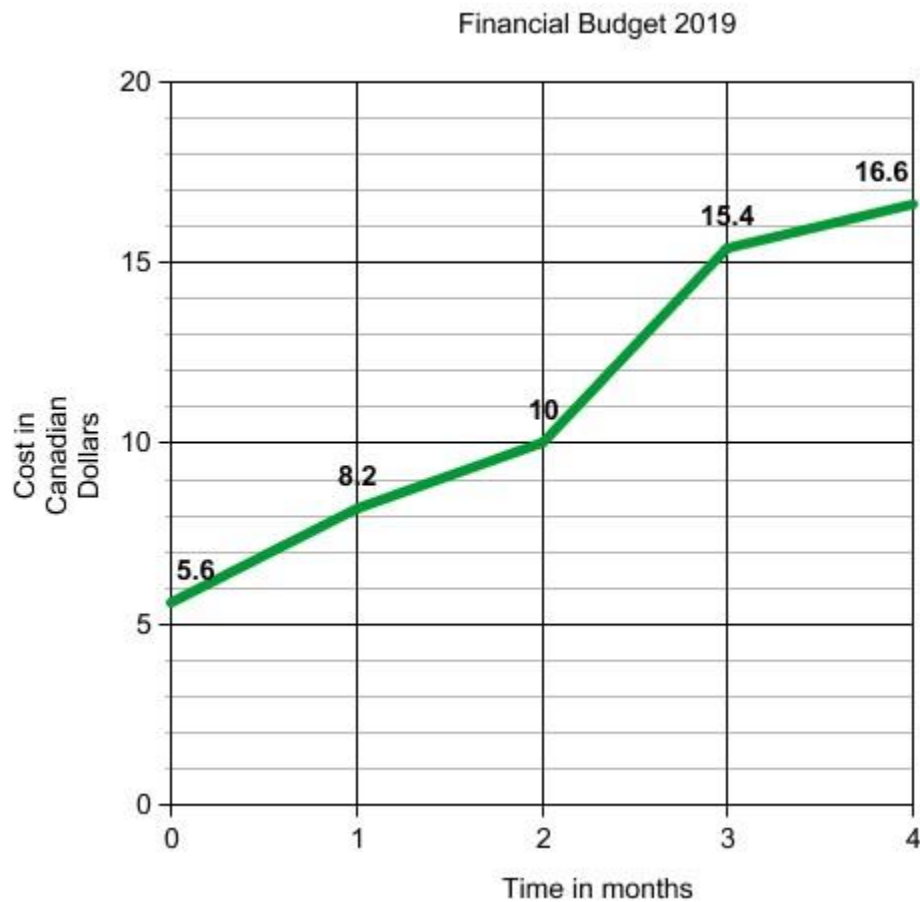
Table 2: Technical Constraints of Brass2Go

	Technical Constraints - Description
1	Implementation of the PIC16F1778
2	Must demonstrate integration of ILS Courses
3	Discrete Components
4	Designed and developed by the assigned team
5	Assembled by the assigned team

2.3. Financial Budget

The Financial Budget for this final project was outlined in the ENEL 300 – Project Rubric and the following table summarizes the financial budget for the development of Brass2Go.

Table 3: Financial Budget for Winter 2019 for the duration of 4 months starting January 07th, 2019 – April 18th, 2019.



Our goal for affordability was achieved as the total cost of parts for this project returned \$16.60 (CAD) calculated on April 17th, 2019.

3. Product Design

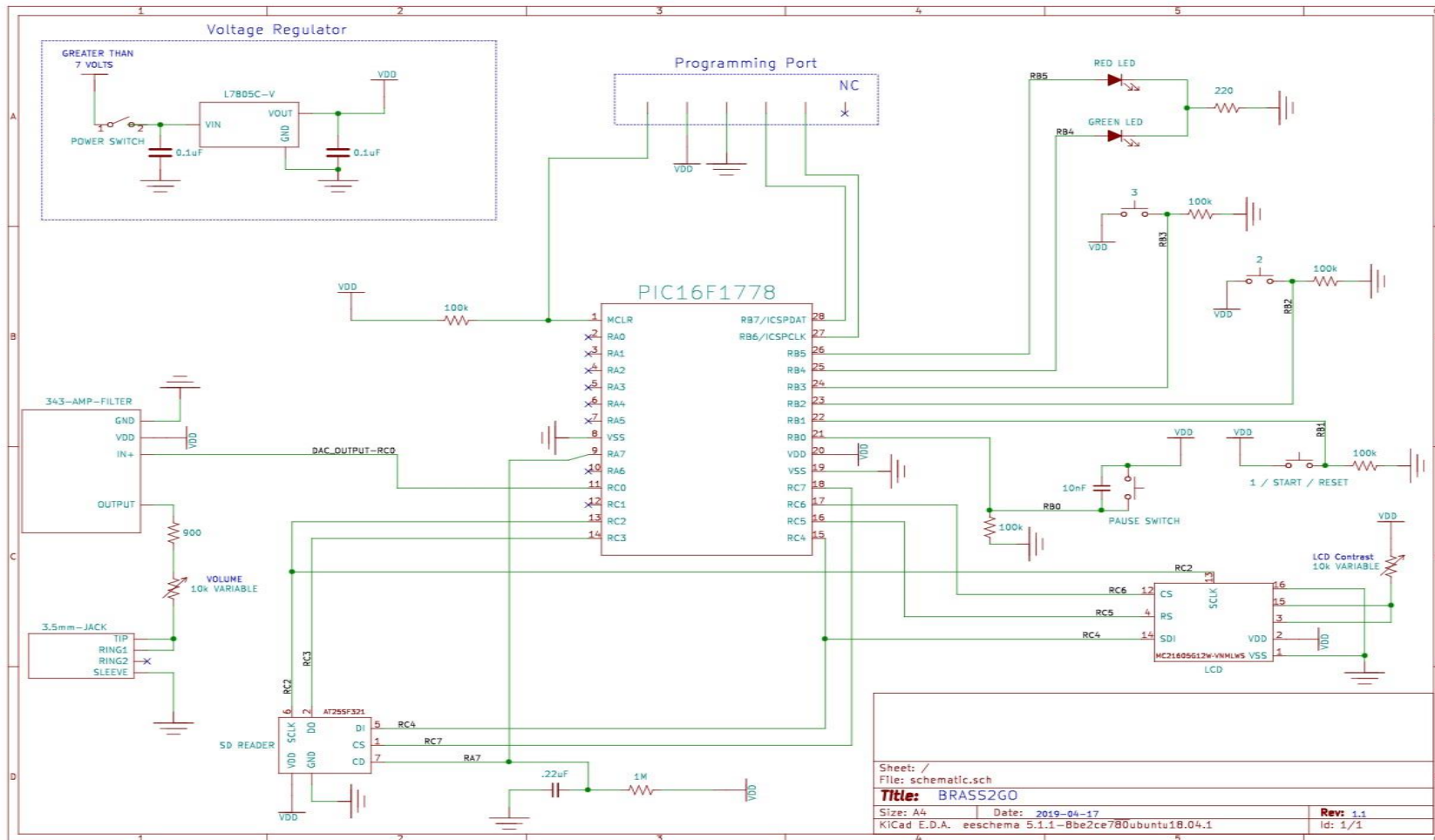
Product Function Description

Brass2Go is a portable device that allows brass musicians to practice their instruments and compositions on-the-go. The device, powered by a PIC16F1778 microcontroller, plays an audio (.wav) file from a microSD card and the user plays the finger sequence along with the song. If the user plays the note incorrectly (i.e. the wrong note and/or at the wrong time) a red LED will light up. In contrast, a green LED will light up if the note is played correctly. Once the file is finished playing, an LCD screen will display the number of notes played incorrectly, and the percent of notes played correctly (e.g. 5/26 wrong -- 80.7% correct). During playback, the user can pause and play.

3.1. Circuit Design

Table 4: Circuit Schematic of Brass2Go featuring the PIC Microcontroller

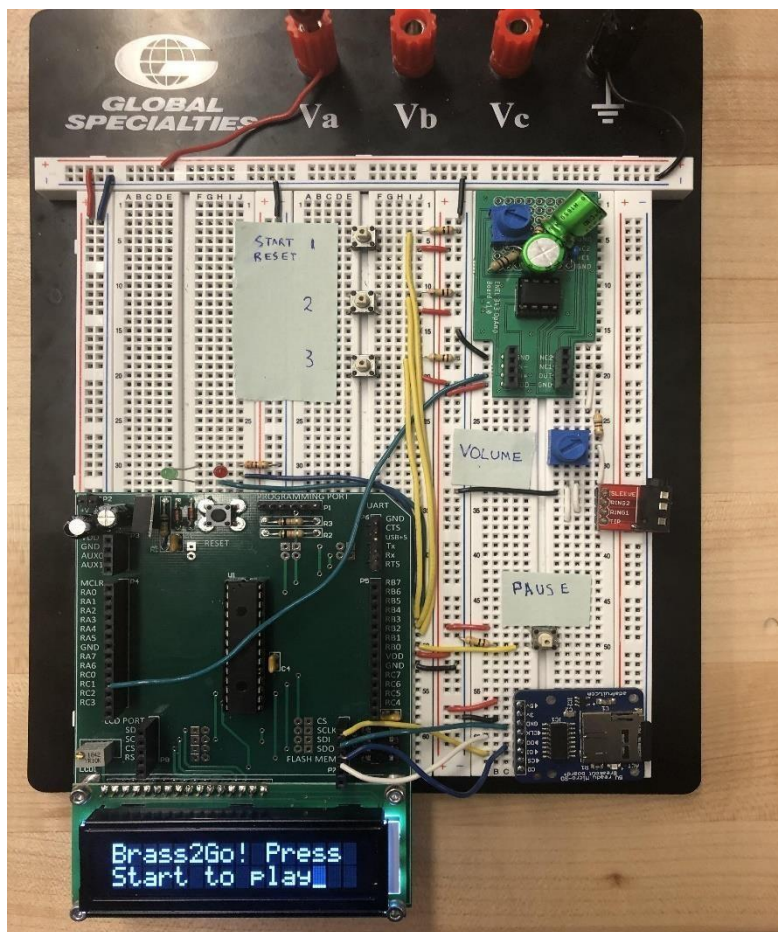
A full resolution PDF of the Schematic can be found on github.com/usmanziak/Brass2Go/tree/master/schematic



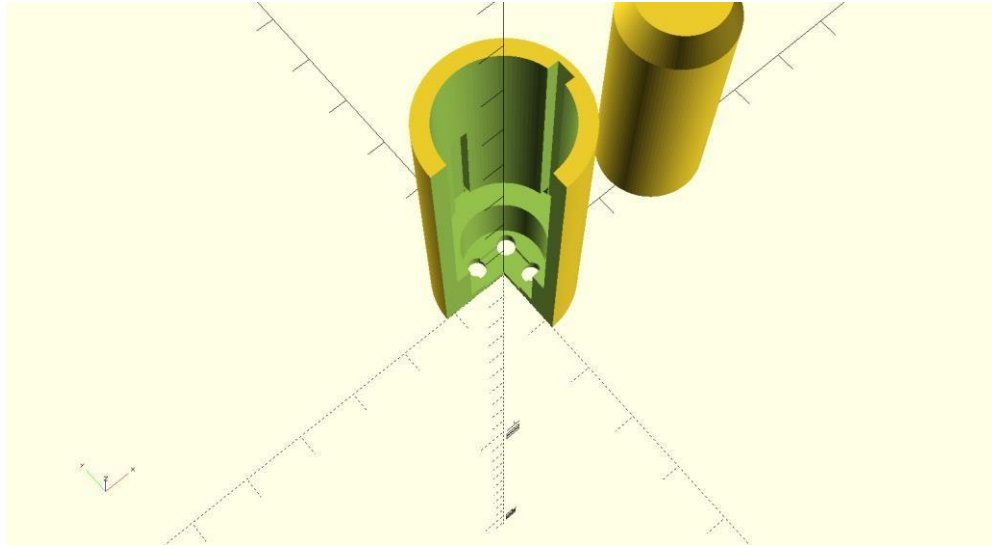
3.2. Hardware/Software Prerequisites

The hardware used for this device is:

- PIC16F1778 Microcontroller
- 2-line LCD Display (Midas MC21605G12W-VNMLWS)
- SPI SD Card Reader (AT25SF321)
- ENEL 343, Circuits II, Audio Amplifier and Filter (TLV4112 Op Amp)
- Red and Green 20mA LEDs
- Switches (MJTP Series)
- 3.5 mm Headphone Adapter
- Resistors (10k Ω for switches, 22 Ω for LEDs, 10k Ω potentiometer and 960 Ω for volume knob)



CAD Designed Trumpet Valves emulating realistic trumpet valve:



Software Prerequisites:

The following software must be installed on your Windows, MacOS, or Linux machine:

- MPLAB X
- XC8 Compiler
- Python 3
- Mido (pip package)
- HxD
- MuseScore (optional)
- Audacity (optional)

MPLAB X and XC8 Compiler

Once installed, you can then compile and program the PIC16F1778 microcontroller with the Brass2Go.X project. This only needs to be done once.

Mido

Mido is a python package that is used by formatter.py to parse MIDI files. Most installations of Python 3 include pip (the Python package installer), but in the event pip is not a recognized command.

HxD

HxD is used to load the formatted .wav file to the SD card.

MuseScore and Audacity

These programs are used to generate the required audio files for formatter.py. If you already have MIDI and .wav files for your composition, these programs are not needed. MuseScore is a convenient open-source platform to convert sheet music and MusicXML files to MIDI and audio (.wav) files. Ensure you are using the correct key, and instrument type in MuseScore when you are generating the MIDI and audio files.

Audacity is an open-source platform to edit and mix audio files. Use Audacity to ensure that the audio (.wav) files that you are using are:

- Mono
- 16-bit PCM
- Sample Rate (Project Rate) = 44100 Hz

3.3. PIC Programming

To access all the code click on the images to go to the respective file on Github, or visit <https://github.com/usmanziak/Brass2Go>.

1. The following code snippet explains the workings of the [formatter.py](#) script:

This python script is designed to encode the correct button presses into the .wav file (audio file) so we can play the audio file from the SD Card with encoding information. This script essentially encodes all the MIDI note data directly into the .wav file, in a format that the PIC expects.

```
61     for sector in range(1, os.path.getsize(filename) // 512):
62         sound_file.seek(sector*512)
63         original_byte = int.from_bytes(sound_file.read(1), "big")
64
65         if sector in sector_list: # IF buttons must be pressed
66             i = sector_list.index(sector)
67             button_bits = note_list[i] << 1
68             new_byte = original_byte & 0xFE | 1 # Set LSB to 1
69             new_byte = new_byte & 0xF1 | button_bits # Set bits 1-3 to button
70         else:
71             new_byte = original_byte & 0xFE | 0 # Set LSB to 0
72
73         sound_file.seek(sector*512) # seek to correct sector
74         sound_file.write(new_byte.to_bytes(1, "big")) # write to .wav file
75     sound_file.close()
```

The above code will check if the button should be pressed or not and set the Lowest Significant Bit to 1 if the button should be pressed and vice versa for setting the LSB to 0 if button not pressed.

2. Main function and Super-loop task

The `void main(void)` function is where the PIC first starts. `Init()` will initialize all required variables and peripherals. Then, the super-loop will run whatever task is next.

```
90 void main(void) {
91     INTCONbits.GIE = 0;
92     init();
93     address = 0;
94
95     TRISA6 = 0;
96     // TRISA7 = 0;
97
98     LCD_SELECT();
99     LCD_DATA_MODE();
100    task_startScreen();
101    LCD_Cmd(LCD_CLS);
102    LCD_DATA_MODE();
103    LCD_Print("Playing!");
104    LCD_DESELECT();
105
106    SD_SELECT();
107    openFile(address);
108    if(channels != 2) samplePending = false;
109    timer_Init(sampRate);
110
111    task = task_playing;
112    while(1) { // main super loop, will execute the function that is pointed to
113                // by task
114        task();
115    }
116 }
```


3. Task_Playing Function

This function is the longest, and most involved function of the Brass2Go program. It populates a buffer of music samples from the SD card to output to the speaker or headphone jack. Every so often, an interrupt will run and output the next sample in the buffer to pin RC0. At the start of each SD card block/sector, it will check if the current trumpet valve state needs to be sampled. If so, it samples the buttons accordingly, and compares it with the expected valve combination. It also checks to see if the user has paused the song, and if the song has ended or not. In either case, the task will switch to `task_paused` or `task_analysis`, respectively.

```
130 void task_playing()
131 {
132     while(1)
133     {
134         if (blockIndex >= 512) { // end of block condition
135
136             DAC_INT(0);
137
138             // Check for end of the file
139
140             if (byteCounter >= dataLength) {
141                 PIE1bits.TMR2IE = 0;
142                 SD_CloseStream();
143                 SD_DESELECT();
144                 task = task_analysis;
145                 return;
146             }
147
148             // Read 4 CRC bytes at the end of the block
149             SPI_POKE();
150             SPI_POKE();
151             SPI_POKE();
152             SPI_POKE();
153
154             DAC_INT(1);
155
156             check_buttons = true;
157             ++address;
158             blockIndex = 0;
159
160             // Check for 0 -> 1 transition of the pause button
161             current_pause = PAUSEBUTTON;
162             if(current_pause && !previous_pause) {
163                 task = task_paused;
164                 return;
165             }
166             previous_pause = current_pause;
167
168             } else {
169                 DAC_INT(0); // disable timer interrupts while accessing buffer_read_index
170                 if (buffer_write_index != buffer_read_index) { // read into the buffer if there's space
171                     DAC_INT(1);
172
173                     if (channels == 1) {
174                         LATA6 = 1;
175
176                         // Read 16 bit sample into sdata_lo/hi
177                         SPI_READ(sdata_lo);
178                         SPI_READ(sdata_hi);
179
180                         //write to the buffer
181                         lbuffer[ buffer_write_index ] = ((sdata_hi << 8) | sdata_lo) - 0x8000;
182
183                         //
184                         LATA6 = 0;
185
186                         byteCounter += 2;
187                         blockIndex += 2;
188
189                         //CHECK IF THE CORRECT BUTTONS ARE PRESSED AND
190                         //ADD TO THE NUMBER OF ERRORS IF IT IS WRONG
191
192                         if(check_buttons) {
193                             check_buttons = false;
194                             first_byte = sdata_lo;
195                             if(first_byte % 2 == 1){ //IF BIT 0 == 1
196
197                                 // note is to be played here, so check which buttons are depressed
198                                 ++total_presses;
199
200                                 if(!Check_Buttons(first_byte))
201                                 {
202                                     // The buttons were wrong
203
204                                     ONRED
205                                     OFFGREEN
206                                     ++number_of_errors;
207                                 }else
208                                 {
209                                     OFFRED
210                                     ONGREEN
211                                 }
212                             }
213                         }
214                     }
215                 }
216             }
```

```
unsigned char Check_Buttons(unsigned char encoded_byte)
//Returns 1 if the buttons that are currently pressed are the buttons that should be
//Pressed (encoded byte)
{
    unsigned char correct_buttons = (0b00001110 & encoded_byte) >> 1;
    if(Buttons_Pressed() == correct_buttons)
        return 1;
    else return 0;
}
```

4. Task_Paused Function

The `task_paused` function simply disables the audio output, then waits for the user to press either the pause button again (to resume playing), or Valve 1 (for a reset of the program). When the user presses the pause button again, the buffer will reset to its initial states, and the current task is switched to `task_playing` to resume audio playback.

```
241 void task_paused()
242 {
243
244     //Disable SD Card SPI interface and enable LCD
245     DAC_INT(0);
246     SD_CloseStream();
247
248     SD_DESELECT();
249
250     LCD_Cmd(LCD_CLS);
251     LCD_Print("Paused");
252     do {
253         previous_pause = current_pause;
254         current_pause = PAUSEBUTTON;
255
256         //RESET
257
258         if (VALVE1)
259         {
260             delay(250);
261             LCD_Cmd(LCD_CLS);
262             RESET();
263         }
264     } while (!(!previous_pause && current_pause));
265     previous_pause = true; // prevent a new pause from being triggered
266
267     LCD_Cmd(LCD_CLS); //clear display
268     LCD_Print("Playing!");
269     LCD_DESELECT();
270
271     SD_SELECT();
272
273     // Re-open SD card at the last address
274     SD_OpenStream(address);
275     //Reset buffer
276     buffer_read_index = 0;
277     buffer_write_index = 1;
278
279     task = task_playing;
280     // delay(750); // Delay playback so user can get ready to play the next note
281     wasPaused = true;
282 }
```

5. Task_analysis Function

The `task_analysis` function calculates and displays the results of the completed playback. It writes the number of errors, and the percentage of notes played correctly by the user to the LCD.

```
284 void task_analysis()
285 {
286     LCD_Cmd(LCD_CLS);
287     char message[34];
288     sprintf(message, "%d/%d wrong\n%.2f%% correct", number_of_errors, total_presses, 100*(float)(total_presses-number_of_errors)/total_presses);
289     LCD_Print(message);
290
291     while(!PORTBbits.RB1);
292
293     // Reset
294     delay(250);
295     LCD_Cmd(LCD_CLS);
296     RESET();
297 }
```

6. The use of interrupts

One interrupt was used in this program. Its main use is to write the next sample in the audio buffer to the audio output pin RC0. It is also used to detect and handle an SD card not being inserted to the device.

```
74 void __interrupt() isr(void) { // modifies buffer_read_index
75     if (IOCAF7) {
76         // CD INTERRUPT
77         INTCONbits.GIE = 0; // disable all further interrupts
78         card_removed();
79     } else {
80         // AUDIO INTERRUPT
81         TMR2IF = 0;
82         unsigned short level = lbuffer[buffer_read_index++];
83         DAC5REFH = (level & 0xff00) >> 8;
84         DAC5REFL = (level & 0x00C0) << 8;
85         DAC5LD = 1;
86         if (buffer_read_index >= BUFFER_SIZE) buffer_read_index = 0;
87     }
}

299 void card_removed() {
300     INTCONbits.GIE = 0; // disable all interrupts
301
302     DAC5REFH = 0; //set the DAC to zero to avoid a weird whine
303     DAC5REFL = 0;
304     DAC5LD = 1;
305
306     delay(5);
307     SD_DESELECT();
308     LCD_Cmd(LCD_CLS);
309     LCD_Print("NO CARD DETECTED\nPlease insert SD");
310
311     delay(500);
312     while(!SD_CD_PIN);
313     RESET();
314 }
```

4. Product Development Cycles

Alpha Phase:

The idea behind the project was to create a learning tool with accuracy output for brass instrument players in order to give them the opportunity to practice their skills on the go. As suggested during the Alpha Phase of the project we launched a basic functional prototype but found several issues with running the LCD, input from the keys and our accuracy algorithm at the same time given limitations with the PIC microcontroller.

Beta Phase:

During the Beta Phase we had to change fundamental functionality of Brass2Go, from the results of our Alpha Phase, the most salient being that we decided on not providing LCD output during the song. However, we successfully tested displaying an accuracy percentage after every song played.

Pilot Phase:

Finally, in the Pilot Phase our group focused on our persona and improved the usability of Brass2Go by creating a portable final product of Brass2Go with features such as:

1. Battery Powered – To make it easier to use on the road.
2. Head Phone Jack – For our persona to play Brass2Go in public.
3. Improved keys – Adds in to the high-quality “feeling” of our product.

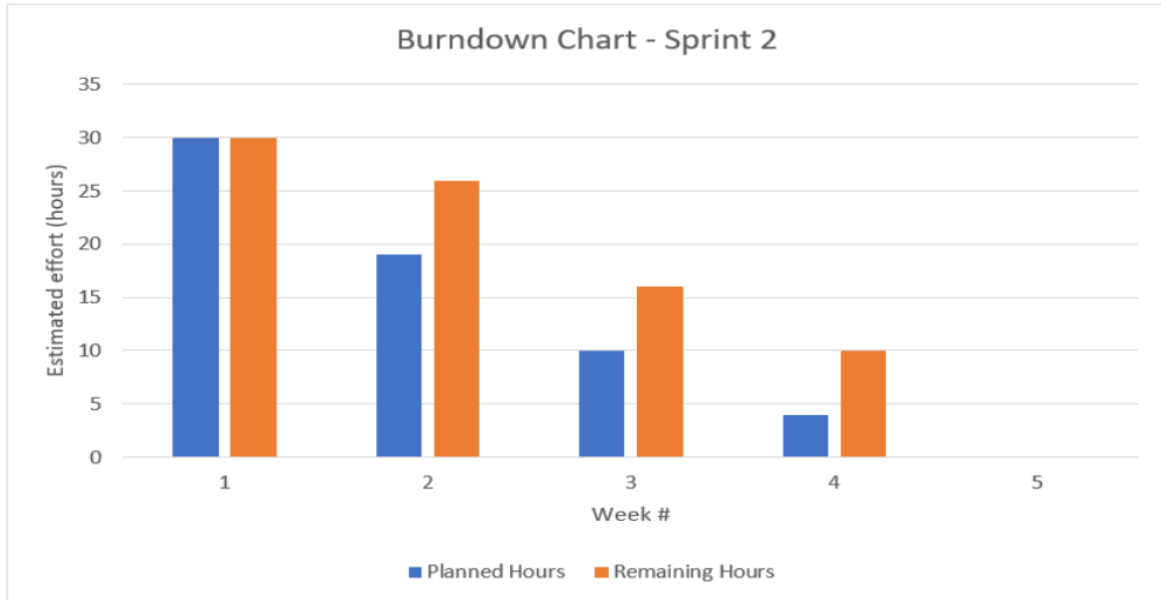
5. Project Management

The development process for Brass2Go was heavily based on the Agile Methodologies:

- In Agile project management, testing is integrated during the cycle, enabling our group to make immediate changes as issues arise. This management technique has significantly improved our execution of Brass2Go.
- Incorporating daily testing and continuous integration allowed faster addressal of issues and in time as the knowledge of Brass2Go was as relevant as possible.
- Moreover, Agile technique also comprises of conducting sprint retrospectives resulting in continuous improvements and increased project control with team transparency.
- Overall, Agile Project Management in ENEL 300 gave us the opportunity to learn about effective project execution with several development cycles and how to incorporate continuous testing.

Attached is a copy of the second sprint – Backlog and Burndown chart for our evidence of efficient team management to deliver this project:

Burndown chart



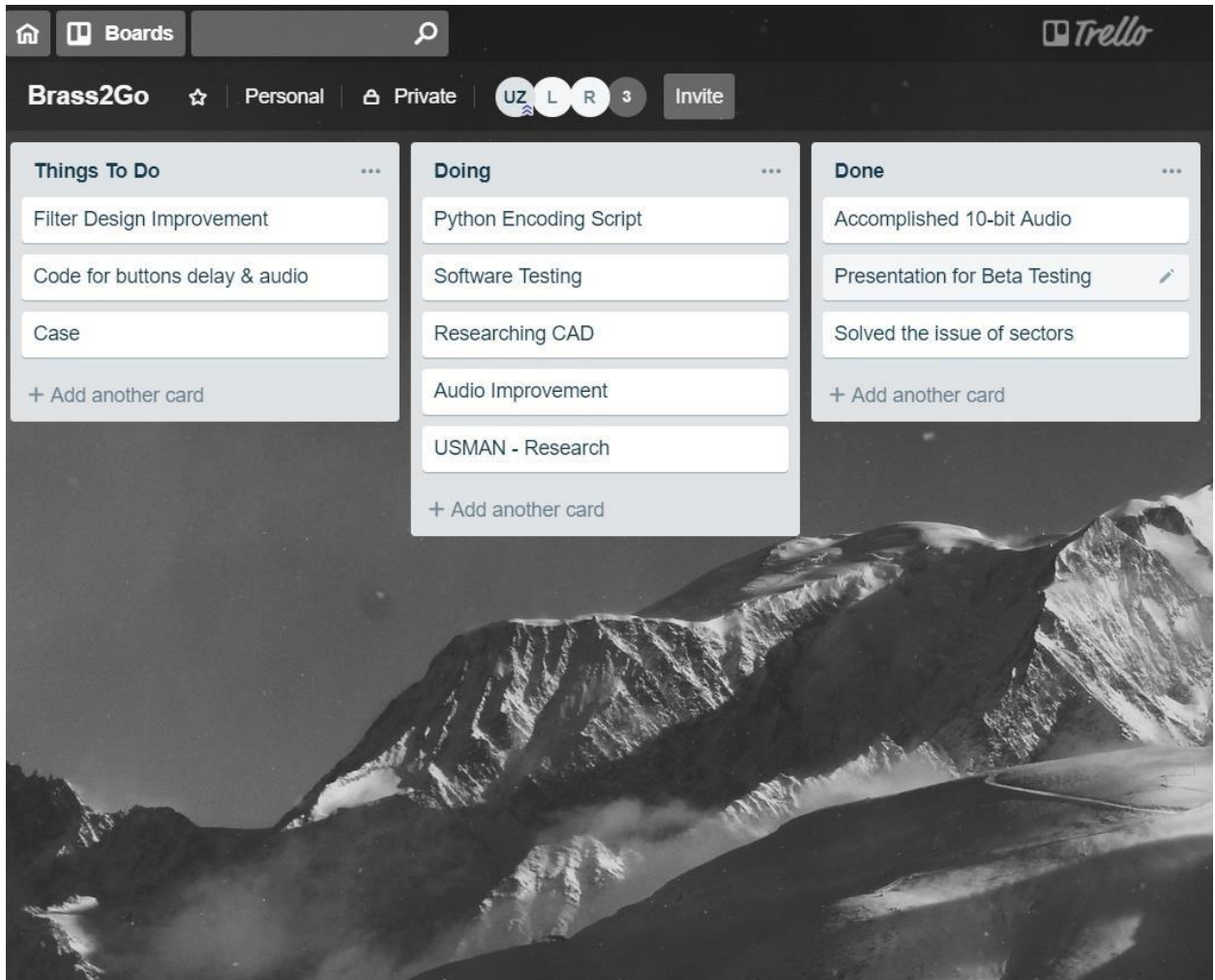
ENEL 300 – ILS February 15 handout

Team name:

Team members:

Sprint 2 - Backlog

Product	Effort Estimate	Responsible team member
Designing a 3D print case	8 Hours	Usman
ENEL 343 Amplifier Lab	3 Hours	All
ENEL 343 Filter Lab	3 Hours	All
PIC Lab 2 – DAC Configuration	8 Hours	All
PIC Lab 3 – Play MP3 files from SD Card	8 Hours	Toshi, Luke
Adding input buttons	2 Hours	Toshi
Research about playing brass instruments	2 Hours	Toshi
Plan out software applications	5 Hours	Luke



6. Validation and Testing

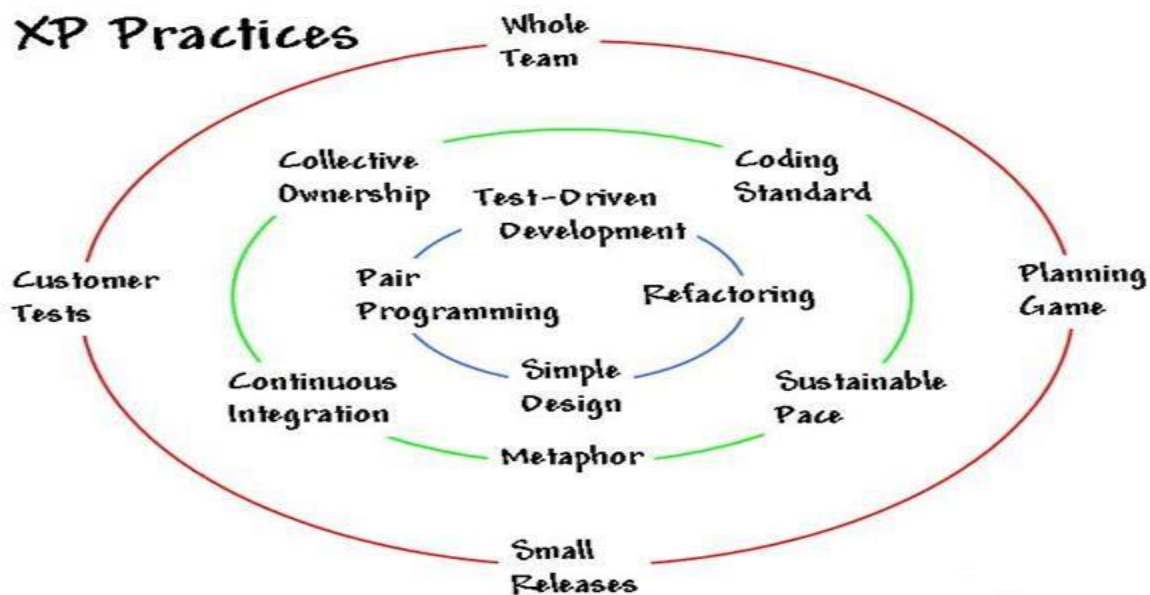
6.1. Hardware Testing

Hardware Testing comprises of continuity testing which essentially tests for solder bridges and other possible issues on the designed perf boards. Moreover, the oscilloscope is a great tool for further testing the output at several connections and specifically the output from the PIC headers.

6.2. Software Testing

We integrated Agile development into our software development process. That includes continuous integration, basically testing your software at every branch merge on Git.

This allowed us to check for errors at every integration by a team member on the master branch. The following diagram displays how Agile Methodologies tie in together:



[6]

7. Appendices

7.1. References

- [1] K. Murari, 'ENEL 361 Professor', University of Calgary, 2019.
- [2] J. Long, 'Teaching Assistant', University of Calgary, 2019. [3] PIC16F1778 Microcontroller, microchip. [online] Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001819B.pdf> [Accessed on: 3/23/2019]
- [4] Author: Steve Fuller, Date updated: 4/17/2019, <https://www.statista.com/topics/1639/music/> [Accessed on: 4/15/2019]
- [5] ENEL 369 LAB Documents, "University of Calgary", 2019
- [6] Image: <https://www.guru99.com/testing-methodology.html#4> [Accessed on: 4/17/2019]

7.2. User Manual

Are you ready to become a professional brass instrument player and take your skills to the next level? Well, look no further as “Brass2Go” is designed for music fanatics like you!

Why was it created?

Brass2Go was inspired by the highly competitive music industry and the lack of a portable learning tool to practice brass instruments. Over 90% of music professionals are considered “unknown” by the public and only less than 1% of all music professionals showcase their talent to people like you and I. Why are we restricted to certain musicians and limiting our creativity?

Thus, Brass2Go is a learning tool for the 90% we would like to cater to and help them improve their skills to compete with the successful 1% musicians and share their talents with the world!

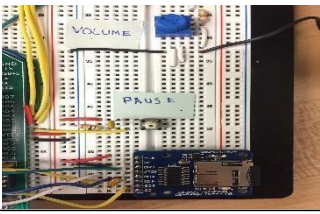
What is Brass2Go and how does it help you improve your skills?

Brass2Go is a portable device that allows brass musicians to practice their instruments and compositions on-the-go. The device, powered by a PIC16F1778 microcontroller, plays an audio (.wav) file from a microSD card and the user plays the finger sequence along with the song. If the user plays the note incorrectly (i.e. the wrong note and/or at the wrong time) a red LED will light up. In contrast, a green LED will light up if the note is played correctly. Once the file is finished playing, an LCD screen will display the number of notes played incorrectly, and the percent of notes played correctly (e.g. 5/26 wrong -- 80.7% correct). During playback, the user can pause and play.

Play Brass2Go in only 5 easy steps

Step 1: Load the SD card.

Use the instructions on the GitHub README document (under Formatting and Loading Files) linked [here](#), or on Page 27.

<p>STEP 2</p> 	<p>Press the switch to turn on your Brass2Go</p>
<p>STEP 3</p> 	<p>Press the first valve to start your desired song stored in the SD Card</p>
<p>STEP 4</p> 	<p>Start your magic and play the trumpet as soon as you hear the song! (Can also PAUSE/PLAY)</p>

STEP 5

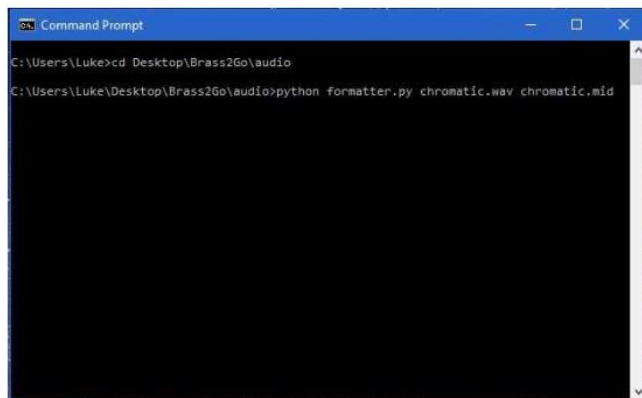


The LCD will output your results at the end of the song.

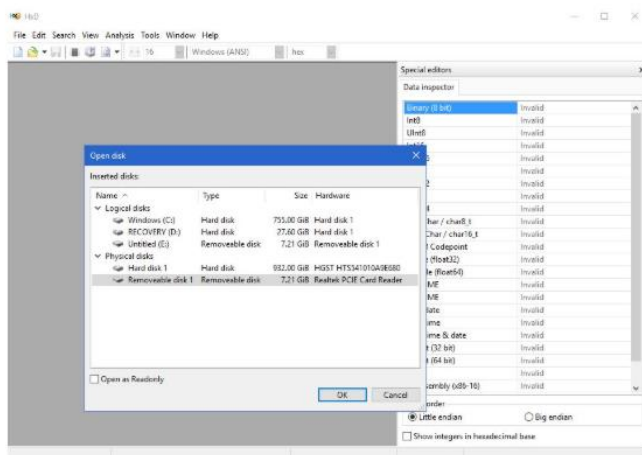
Formatting and Loading Files

Assuming all prerequisite software is installed on your machine, and that you have correctly created your MIDI (.mid) and audio (.wav) files, follow the following steps to add it to the SD card. As an example, we will format and upload `chromatic.mid` and `chromatic.wav` (located in the `~\Desktop\Brass2Go\audio` folder) on a Windows 10 machine.

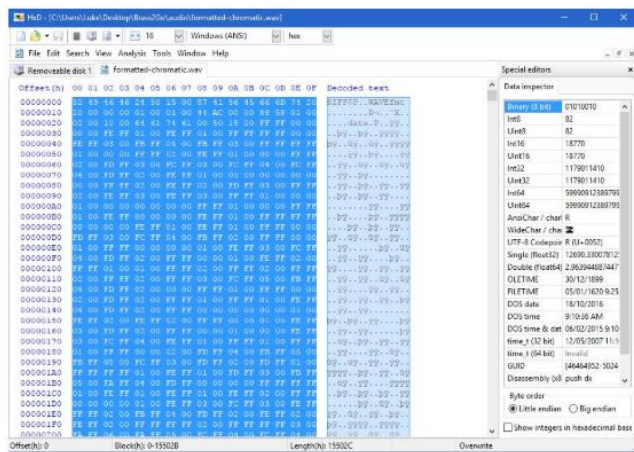
1. Add the .mid and .wav files to the `\Brass2Go\audio` folder, and navigate to that directory using the command `cd 'PATH TO Brass2Go\audio'`
2. Run the Python 3 script using the command `python formatter.py audio_file.wav MIDI_file.mid`



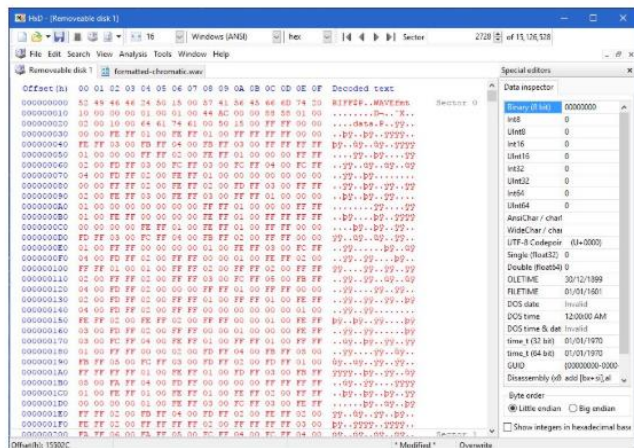
3. Open HxD 'As Administrator' and open the 'Tools' menu, and press 'Open Disk' (Windows Keyboard Shortcut `Ctrl+Shift-D`), and open the SD card. Make sure you de-select the 'Open as Readonly' option.



4. Open the file that was formatted by the Python script (`formatted-chromatic.wav` in this example). Copy all the data to the clipboard by using the shortcuts `Ctrl-A` then `Ctrl-C`.



5. Copy all the data into the SD card by using the shortcut `Ctrl-B`, then save it to the card by using the shortcut `Ctrl-S`.



The file is now successfully on the SD card. Place the card into the card reader on the device. Then, power on the device, and it should function correctly.