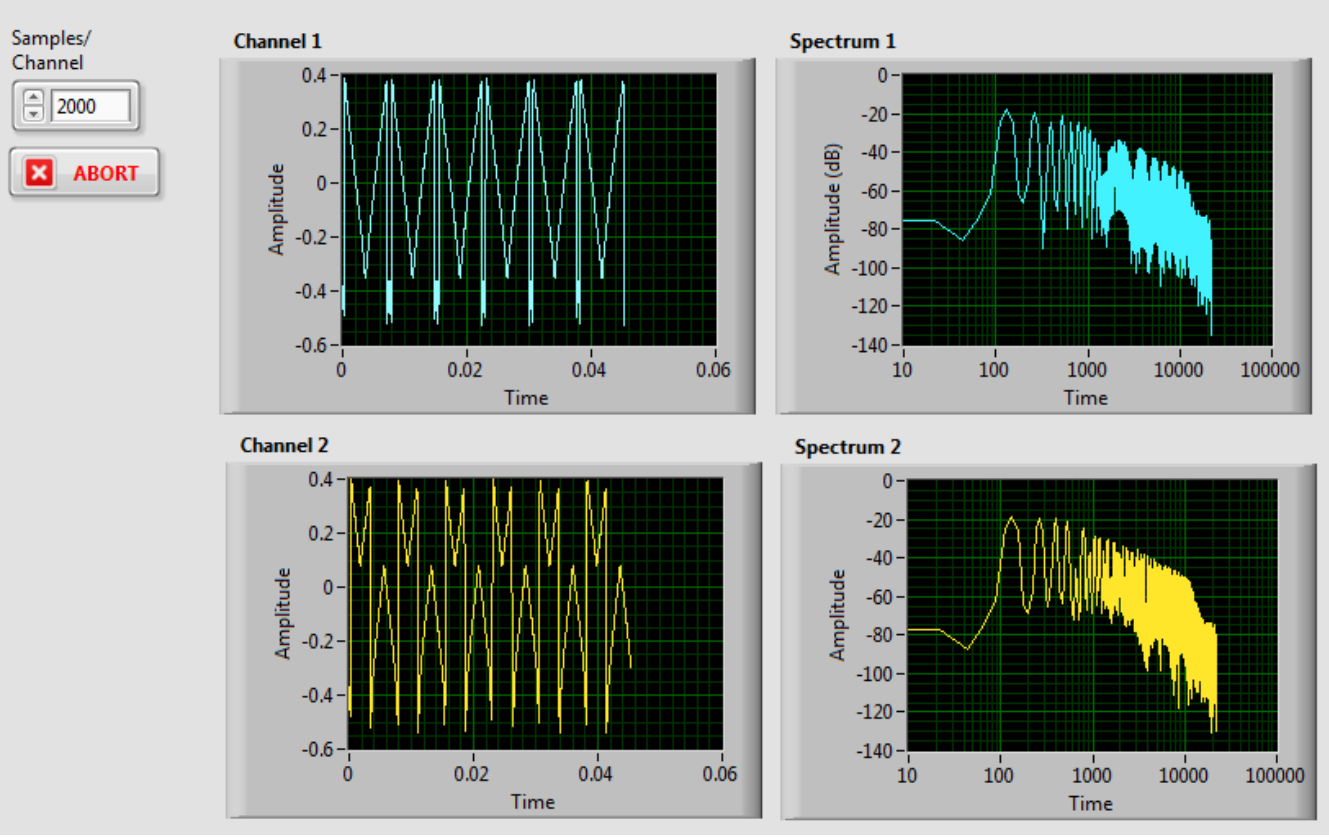
Physics 306L: LabView Assignment 8
Audio Spectrum Analyzer

This assignment uses LabView to access the sound card on your computer to make a real-time display of the stereo signals produced by an audio (.wav) file. The audio file is opened, parsed inside a For Loop, and then closed. Resource management is important to ensure a smooth playback.

The VI is built around a state-machine, which was the subject of Assignment 7. Use the design template for a standard state-machine and configure its type-def constant for four states: Setup, Play, Open File, and Stop. On the Front Panel, place four waveform graphs: two graphs will display the time output for each channel and two will display their Fourier transforms (spectra). Also place a numeric control (I32 representation) for the Samples/Channel and a Boolean controls to abort playing of the current .wav file as shown below.



Best playback is obtained by eliminating the Wait function inside the While Loop. The CPU can then devote all its available resources to audio operations.

Code the Setup state first. Select Graphics & Sound: Sound: Output: Sound Output Configure.vi and place it in this state. Wire the Samples/Channel control (I32) to the **number of samples/ch** terminal. Create a constant on the **sample mode** terminal and

select “Continuous Samples”. Create a constant on the **sound format** input terminal and set the cluster values as follows: sample rate = 44100, number of channels = 2, and bits per sample = 16. These setting should work for modern sound cards, but you may have to experiment depending on the hardware you have on your computer. The configure operation will create a **Task ID** that is needed in the Play state. Add a shift-register on the While Loop perimeter and connect the **Task ID** to it.

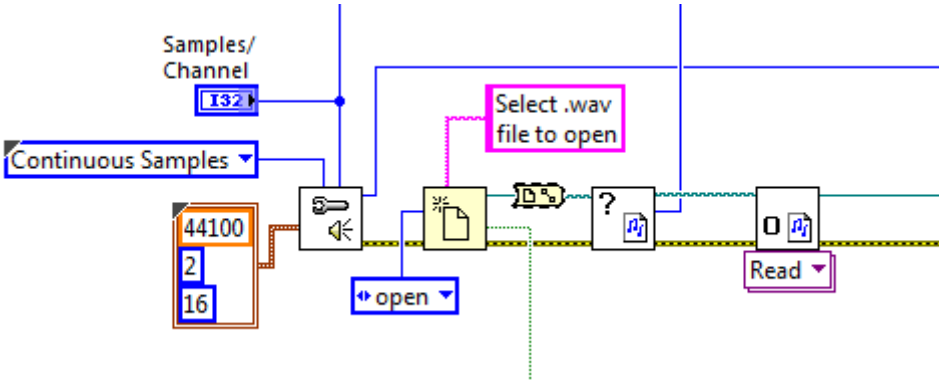
In this Setup state, the input .wav file is specified. From the File I/O palette select Open/Create/Replace File and set **operation** to open. Add a text string to the **prompt** terminal instructing the user to browse for a .wav file to play. The **refnum out** data must be converted to a file path; use the Refnum to Path function located in the Advanced File Functions sub-palette for this. In the Sound palette, locate Files: Info and place it in the Setup state, then wire the file path to its **path** input terminal. We are interested in the total number of samples in the selected .wav file. Divide the **total number of samples/ch** output by the Samples/Channel input control. This sets the number of times the For Loop in the Play state will run. Round this data to an integer and convert it to I32 representation. Make this data available in the Play state by placing it in a third shift-register.

The file path must be converted to a sound file refnum. Make this conversion with Sound File Open.vi and pass this refnum into a fourth shift-register. This is a polymorphic VI that can be configured for Read or Write. Use the default Read operation. You should now have four shift-registers on your state-machine. It's a good idea to use the text tool to label the individual shift-registers outside the While Loop so you know what data they contain.

Another way to make data available in other states is with a Local Variable. In the Block Diagram, right-click on the Samples/Channel control and create a Local Variable. Place it outside the Case Structure. It will be wired in the Play state.

It is important to enforce the execution order to first configure the sound card, then open the .wav file, read its size, and generate a sound file refnum. Use the error cluster for this. The user has the option of pressing the Cancel button if prompted to open a .wav file. If this happens, the VI should be stopped. The **cancelled** terminal of the open file operation will produce a Boolean TRUE output. This flag is used to take the program to the Stop state. If the **cancelled** output is instead FALSE (this means a .wav file has been specified and opened) the next state should be Play. If an error occurs, the VI should also be stopped. Add some code (you will need an OR gate) to make this selection. (Note: The error cluster does not have to be unbundled to access the status Boolean. It can be wired directly to any Boolean terminal.) Pass the error cluster to other states with a fifth shift-register on the While Loop.

Part of the code for the Setup state should look similar to this:



Now go to the Play state and insert a For Loop. Wire the **Count** terminal to the value calculated in the Setup state; this number is available on the corresponding shift-register. Right-click on the edge of the For Loop and select "Conditional Terminal". Wire the Abort button Boolean control to it. Notice how the **Count** terminal changes to indicate the presence of the conditional stop.

Inside the For Loop, the .wav file data will be parsed, played, and analyzed. In the Sound sub-palette, select the Sound File Read.vi and place it in the For Loop. (Note: This is not the same VI used in the setup state.) Using the appropriate shift-register, wire the refnum data from the opened .wav file to the **sound file refnum** input terminal. Create a constant on **position mode** and select "Absolute".

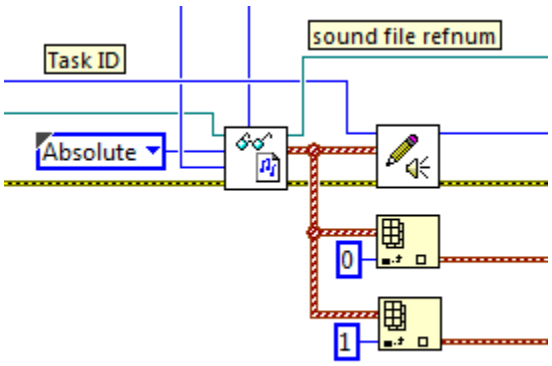
Find the Local Variable for Samples/Channel created in the Setup state. Right-click and select "Change to Read". Place it outside the For Loop and wire it to the **number of samples/ch** terminal. The Local Variable makes an explicit connection to the control terminal in the Setup state and reads it.

Add a shift-register to the For Loop and use it to increment the read position on the .wav file for each iteration of the For Loop. Simply sum the shift-register by the constant value of the Samples/Channel Local Variable with each iteration. Wire the incremented count to the **position offset** terminal of the read function. Don't forget to initialize the shift-register to 0 as an I32 data type.

The goal is to simultaneously play the file through the sound card and display the waveforms for each channel in real time. The parsed sound data is present on the **data** terminal. There are up to two channels of data, so the output is a 2-D array. To play the parsed data, place Sound Output Write.vi in the For Loop and wire the 2-D data array to the **data** terminal. The **task ID** is on the corresponding shift-register from the Setup state. To prevent errors on subsequent plays, the sound buffer must be cleared when the

For Loop finishes. Place the Sound Output Clear.vi *outside* the For Loop; wire **task ID out** from Sound Output Write to the **task ID** input terminal on the clear function. You will need to disable indexing on the For Loop tunnel. In a similar manner, close the sound file using the Sound File Close.vi placed *outside* the For Loop. Connect the **sound file refnum** to its input terminal. The error cluster can be connected between the read, write, and clear operations to help troubleshoot potential problems.

The waveform data from the two channels is in a 2-D array; each channel should be displayed separately *inside* the For Loop. To separate them, use the Array: Index Array function, one for each channel. Wire the **data** output of the Sound File Read in parallel to two copies of Index Array. The 2-D array data is wired to each **n-dimensional array** input terminal. Wire a constant 0 on one **index** input terminal and 1 on the second **index** input; see below:



The two output waveforms can now be connected to the corresponding waveform graphs on the Front Panel.

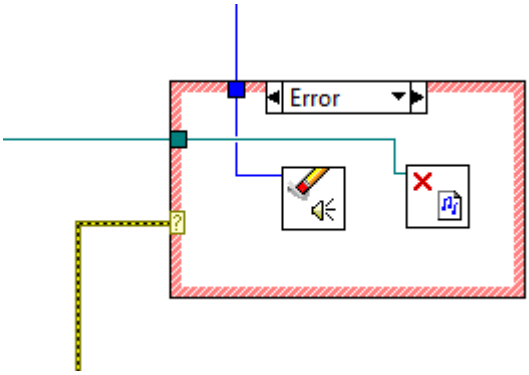
The Fourier power spectra of both waveforms is easily obtained with a built-in LabView function. Go to Waveform: Analog Wfm: Measurements: FFT Power Spectrum & PSD and place two copies inside the For Loop. Wire one waveform to the **time signal** input terminal of each. Connect the **Power Spectrum/PSD** output terminal to the corresponding waveform graphs and set the **db On (F)** terminals for both to TRUE. This will display the y-axis amplitude as a dB log-scale.

When it finishes, the Play state should transition to the Open File state unless an error occurs. Use the Select function to monitor the status of the error cluster and transition to the Stop case if necessary.

Next configure the Open File state with a Two Button Dialog. Wire the message with the string “Open another .wav file?” or something similar. Wire a string to the Cancel terminal that says Stop. The Boolean output of this dialog function indicates which button was pressed. If TRUE, transition to the Setup case. If FALSE, transition to the Stop case.

The Stop state is the only place where the VI can be stopped. Pass a Boolean constant out of it to the conditional terminal of the main While Loop. The other 3 states should have opposite logic. The state that exits the Stop case is meaningless (it will never be reached), but it can be wired to Stop.

If the other states produce an error while one or more resources are open, simply stopping the While Loop will not close them. Leaving hardware in an uncertain state is not good programming practice. This can be addressed as follows: Duplicate the resource close operations from the Play state and place them inside a Case Structure in the Stop state. The error cluster and refnums are accessed at their corresponding shift-registers; the error cluster is wired to the selection terminal as shown:



If the error status is TRUE, these resources will be forced closed before the VI stops. The error cluster shift-register output terminal should be read with a Simple Error Handler as usual.

The VI will not function if all tunnels exiting the case structure are unwired. If you are sure no data will be needed in the shift-registers in subsequent frames, you can right-click on the tunnel and select “Use Default if Unwired”. It's usually safer, however, to explicitly wire all tunnels leaving the case structure. This decision is up to the programmer.

Test your VI with sample .wav files on the class website. It may be useful to display the frequency axes with a log scale and limit the range to two decades: 100—10 kHz. You should experiment with the waveform Samples/Channel size. If it's too small the playback will sound choppy. If it's too long, it will be difficult to see meaningful waveform data, especially if the music transitions rapidly. This will depend on the hardware of the machine on which the VI is run.