<u>Music Box</u>

By Elina Sendonaris and Phoebe Piercy

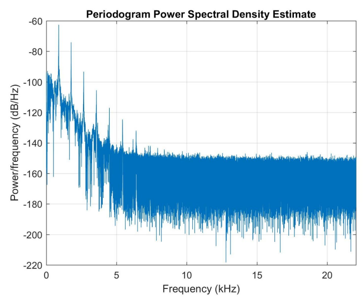
<u>Overview</u>

Our vision is to create some kind of ‘black box’, into which you would feed an audio file, and get out some information about the parameters of the music, which would then be displayed, via VGA, on a screen.

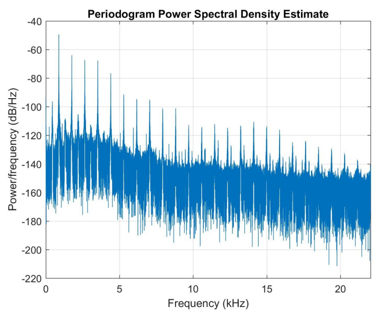
<u>Detailed Functionality</u>

- **Tempo Detection**
  - By passing the audio signal through a low pass filter, in order to isolate the low frequency drum beats from the melody, we would then transform these into a discrete signal of beats and analyse the resulting waveform in the time domain. By calculating the frequency of the peaks, we should be able to output a tempo for the music.
- **Key detection**
  - By passing the signal through an FFT and analysing the frequency spikes, we can map the most common frequencies to their corresponding notes (taking harmonics into account). From this, we should be able to discount accidentals, as they should occur with a much lower frequency, and so detect which sharps and flats are being used, and, from this, isolate the key.
- **Octave detection**
  - Through looking at the highest power spikes at the lowest frequencies, we should be able to identify the fundamental frequency, and thus gain some information as to what octave the music fits into, and display on a sliding scale as to whether it is a lower-pitched piece or a higher-pitched piece.
- **Instrument detection**
  - By feeding our ‘magic box’ an audio file containing a single instrument, we should be able to extract some information about the timbre of the instruments, and thus broadly categorise it as a string, wind or vocal instrument. This is done through the harmonics, as instruments with more ‘husky’ timbres, such as saxophones, have a lot more harmonics than more ‘clear’ instruments, such as pianos. We have done some preliminary research on this and, using MATLAB to display the FFT, have found it fairly straightforward to distinguish a piano and a saxophone, for example. (see below)

PIANO A5



SAXOPHONE A5



- **Chord Progression Detection**
    - This is a stretch goal, as we are currently unsure as to how effectively we are going to be able to isolate the fundamentals from the harmonics. Ideally, however, we would be able to feed a simple ballad (with beat) into the ‘Music Box’, and tell it the time signature. This would allow it to calculate the tempo from the beat, as discussed above, and therefore calculate the number of clock cycles between theoretical chord changes, giving us a sample rate. From here, we would do an FFT calculation for each sample/chord, and use this to extract the fundamental frequencies. We have tested this with MATLAB and, from a simple root position chord, we were able to identify the fundamental frequencies corresponding to the notes in the given chord. It might be that we are only able to do this for very simple pieces of music, so that is or goal and, if we can generalise it to more complex music, then that is a bonus.
- **Display:**
    - We would like to implement multiple modes for the display.
        - Basic info - Key, tempo, detected frequencies, notes.
        - Instrument detection Info
        - Score - Draw staves with the chord progressions detected [STRETCH GOAL]

Modules

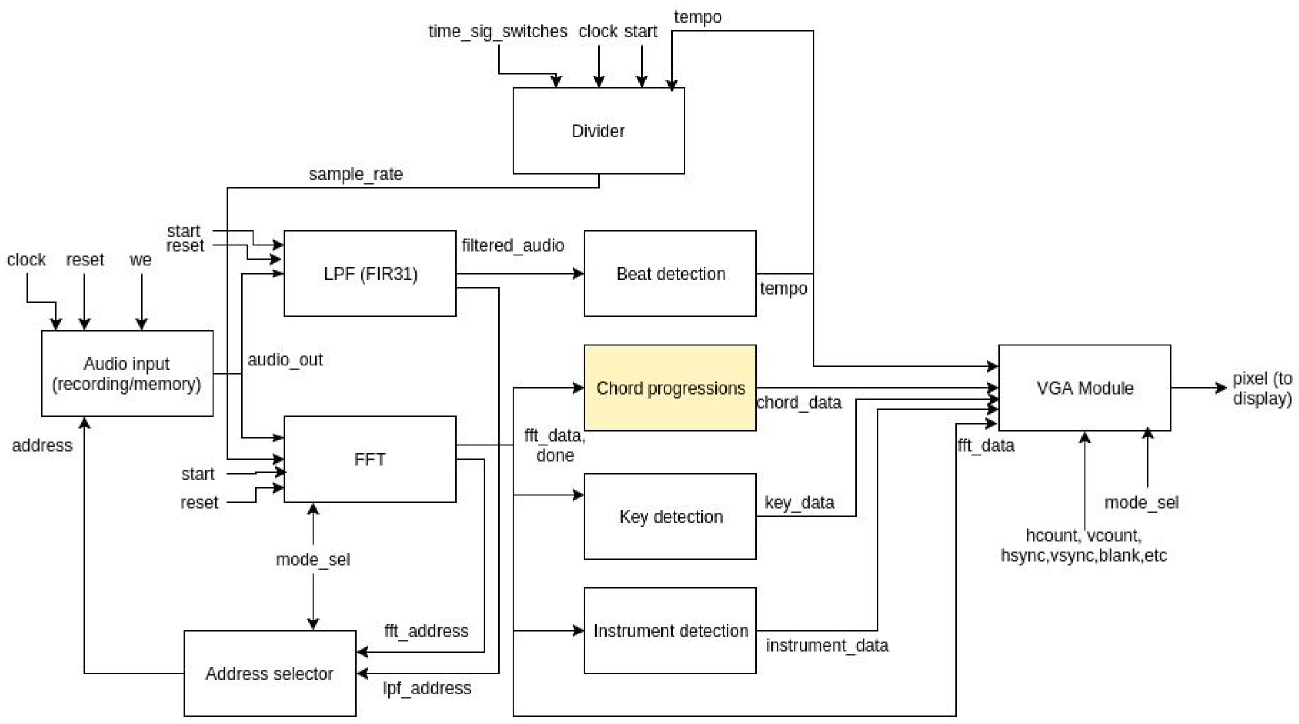
- **Debouncer:**
    - We will be using switches to input time signature information, and to change modes, so we will need a debouncer module.
    - *Complexity:* low
    - *Person*: Phoebe
    - *Testing*: Pretty established
    - *Inputs*: clock, switch,. *Outputs*: debounced_switch.
- **Divider:**
    - In order to calculate sample rates, we will need a divider.
    - *Complexity*: Low. Will need to be generalised enough to output different sample rates
    - *Person*: Elina
    - *Testing*: Input clock and time sig and ensure we can get the right signal out by looking at the waveforms on the simulator.
    - *Inputs*: clock, start, timesig_switches, tempo. *Outputs*: sample_rate
- **Memory**:
    - BRAM to store and read the audio data.
    - *Complexity*: Low. Needs to be big enough to store necessary data
    - *Person*: Elina
    - *Testing*: Use playback to test how much we can store
    - *Inputs*: clock, reset, address, we, audio_in, *Outputs:* audio_out
- **Address selector**
    - Choose which module will access memory, based on mode we’re in

    - *Complexity*: Low. Just has to give the memory the correct address based on outputs of LPF and FFT address and the mode the system is in
    - *Person*: Elina
    - *Testing*: Hook up logic analyzer, and see if the value of "address" changes with changing the mode, based on what lpf_address and fft_address are.
    - *Inputs:* lpf_address, fft_address, mode_sel. *Outputs*: address
- **LPF**
    - To isolate the beat from the melody for the sake of tempo detection
    - *Complexity*: Medium. Needs to be finely tuned to isolate correct frequencies and so should be tested with different drum beats.
    - *Person*: Phoebe
    - *Testing*: Test different results with different values and beats
    - *Inputs*: clock, reset, start, audio_out *Outputs*: filtered_audio,lpf_address
- **FFT Calculator**
    - *Complexity:* Complex. Most arithmetic-heavy module. Will need to be able to pause other operations til this is finished
    - *Person*: Phoebe
    - *Testing*: Display the output after some kind of input
    - *Inputs*: clock, sample_rate, start, reset, audio_out. *Outputs:* fft_data, fft_address, done
- **Peak detection**
    - *Complexity:* Medium. Will need some algorithm work, and should be very general because it will be used by most of the other modules.
    - *Person:*
    - *Testing*: Input an impulse train and ensure it detects the locations of the peaks
    - *Inputs*: clock, start, reset, data_in. *Outputs*: peaks (an array of the locations of peaks)
- **Beat Calculator**
    - *Complexity:* Medium. Needs to be filtered well enough to reliably only have peaks on the beat.
    - *Person*: Elina
    - *Testing*: Input a simple beat and ensure tempo is correctly calculated
    - *Inputs:* clock, start, filtered_audio. *Outputs:* tempo
- **Chord Progression Calculator (stretch goal)**
    - *Complexity:* Complex. Needs to be able to isolate and remove different harmonics to identify the fundamentals and thus the chord. Same basic ideas as key calculation, i.e. looking at peaks of FFT and identifying the notes.
    - *Person*: Elina
    - *Testing*: Input simple chord progression and ensure it outputs correct values. Test with different chord positions and complementary major/minor chords to test for robustness.
    - *Inputs:* clock, sample_rate, FFT_data. *Outputs:* chord_data
- **Key Calculator**

    - *Complexity:* Medium. Needs to be able to identify harmonics and accidentals with some accuracy, although the nature of keys means we don't need to remove these harmonics.
    - *Person*: Elina
    - *Testing*: Test with simple melodic lines, then with accidentals.
    - *Inputs:* clock, key_start, FFT_data, *Outputs:* key_data
- **Instrument Calculator**
    - *Complexity:* Medium. Needs to be able to identify number of harmonics and thus infer instrument timbre and type from this.
    - *Person*: Phoebe
    - *Testing*: Input single instruments across different pitches and test for correct instrument detection.
    - *Inputs:* clock, instrument_start, FFT_data, *Outputs:* instrument_data
- **VGA Module**
    - *Complexity:* Complex. Needs to be able to display the data in a readable format and, if stretch goals are achieved, display complex diagrams.
    - *Person*: Both.
    - *Testing*: Use a testbench to give it signals and ensure it displays the expected outputs
    - *Inputs:* clock, VGA_switches, FFT_data, tempo, chord_data, key_data, instrument_data, hcount, vcount etc etc. *Outputs:* pixels

A lot of our testing is going to involve feeding known audio samples into the modules and seeing what results we get vs what we expect. To this end we will be using music notation software, such as Sibelius, to create our own simple audio files with the targeted music feature in isolation (e.g. simple chord progressions, simple drum beats) to ensure the basics are possible before we feed it more complicated music.

Block Diagram



time_sig_switches
clock
start
tempo
Divider
sample_rate
start
reset
clock
reset
we
LPF (FIR31)
filtered_audio
Beat detection
tempo
Audio input (recording/memory)
audio_out
Chord progressions
chord_data
VGA Module
pixel (to display)
address
FFT
fft_data, done
fft_data
start
reset
Key detection
key_data
mode_sel
hcount, vcount, hsync,vsync,blank,etc
mode_sel
fft_address
Address selector
lpf_address
Instrument detection
instrument_data