Beep Boop 9000

Project Goals

The basic function of our suite will be an audio looper. The GUI will allow the user to record up to 8 different tracks of approximately 30 seconds in length. The user can then choose to play any subset of the tracks that have been recorded while either just listening or playing additional sound over the top of the playback.

The learning synthesizer will work by listening to a note from an instrument and using Fourier analysis to extract coefficients corresponding to each of the relevant frequencies. The user can save different sounds and then select which instrument to playback each track as. The user can also select the instrument to use for live playback.

A stretch goal will be to add a drum kit feature to the system. This could come in the form of several preloaded drum tracks that allow the user to select a genre and a tempo, or we could allow the user to craft his or her own drum beat. The latter option, however, would likely require more extensive work on the GUI side of things.

The Main State Machine

This module will likely require some contribution from each of us, as it acts as a bridge between the GUI and the data management sections of the project. The module will take as inputs the track, action, name, and instrument from the GUI, and then pass these through to the data management module. It will activate modules according to the input from the GUI and keep track of the internal state of the system.

The GUI

Tyler will design and implement a GUI for the project. The team has discussed multiple basic functions that the GUI will possess, but it is likely that these may change slightly or expand as it becomes more clear what the bounds of this project are and the capabilities that are possible. The basic functionalities are: tracks, instruments, recording, active playback, looping, and learning. There will be eight tracks on-screen that the user will be able to interact with. The user can record to an individual track, mute a track (playback includes all tracks that are not muted), select the instrument that the track will sound like, name a track, and delete a track. Tracks allow the user to have eight possible layers that they can playback together to create a song but are all edited separately. Tracks combined with fixed-length loops may also make data storage easier since we can assign constant locations for each track. How the team implements different instruments

is addressed further below, but for the GUI, there will be a selection for each track for the desired instrument that the recorded notes will be played back through. Learning is the process of listening to a new instrument, saving it, naming it, and allowing users to now select it as one of the instruments for playback. This is also further addressed below. There will be a learning button on screen (or perhaps mapped to a button) that will provide instruction on what to do and when. Recording is exactly what it sounds like. There will be instructions and a button on screen for recording too. As of now, there will be a fixed recording window that the user can then delete or overwrite if they did not like it, instead of recording for an unspecified amount of time and selecting a window of that to keep. After recording to a track, the user will have the option of naming it, which will be used as the name of the track. Looping (normal playback) will also have a button on screen. Looping will loop all tracks that are not muted until the user pauses looping. Active playback is a reach goal as of right now, but the team is somewhat confident that it will make it into the final design. Active playback involves real-time transformation of input sound into a chosen instrument. This mode will continue to be on until the user selects to stop it. Most of the options will be accessed by using the buttons. Left, right, up, and down will be used to move between options, and the center button will be used to select or confirm an option. This will be sufficient for most actions the user needs to do, but it may be necessary to also install a potentiometer knob or something similar to address reach-goal actions like volume control or editing. Also, one interesting idea for inputting text to name tracks and instruments is to use the switches to input values 0-25 for the 26 letters of the alphabet.

The Fourier Analysis

Brandon will primarily be working on this section of the project. The Fourier analysis has two main functions. The first is to identify the fundamental frequency of the note being played. This is used in the active conversion of instruments and track recording as well as in the learning. In the active conversion and track recording, this is the final step of the processing and the identified fundamental frequency and associated intensity are passed to the synthesis stage or the storage manager respectively. The second function is learning. The learning flow will produce a set of normalized coefficients representing the timbre for the recorded instrument. In the learning flow, the fundamental frequency and intensity are passed with the FFT data to the Learner module. This module then extracts the coefficients of the harmonics of the given frequency from the FFT data. The module normalizes these coefficients and sends them to the data manager.

The FFT will be performed on data that has been passed through a low pass FIR filter to limit aliasing. Because the information being extracted is going to be used to produce new waveforms, the FFT frequency can be much lower than the sample rate. This will allow vivado to create a more compact IP module saving resources. We will start testing at an FFT frequency of 1kHz. If allowed by resource constraint, we believe this frequency will be responsive enough to intensity changes and note changes in order to faithfully reproduce dynamics and pitch.

The first version of the fundamental frequency finder will just be finding the frequency with the largest amplitude. More sophisticated algorithms may be developed and deployed if performance is unacceptable.

The Synthesizer

Matt will be handling this section of our project. The synthesizer will use the sound characteristics extracted using harmonic analysis to convert a fundamental frequency to the instrument or sound that the user has selected. One of the challenges lies in determining the best way to reproduce the sound of an instrument as well as how accurately we can reproduce the desired sound. We are thinking about using two possible techniques for carrying out this task: a sine-wave lookup table and an inverse Fourier transform. We will experiment with the two methods in order to determine which might be easier to implement. Assuming that the Fourier analysis can be completed without excessive delay, we are hoping to use an FFT followed by frequency coefficient scaling and then an IFFT to convert the sound to the instrument of choice. However, there are a couple of different challenges that we foresee in doing this. The main concern is the delay time resulting from the FFT and IFFT, and another concern is whether this process will allow us to reproduce sounds realistically.

The synthesizer side of the operation will consist of a few steps. The frequency generator module will take in an input fundamental frequency, an amplitude corresponding to the volume, and an instrument's frequency mapping. Using the instrument's frequency coefficients, the module will add these frequencies in correct proportion to the fundamental frequency in order to preserve the volume of the sound while changing the character of the sound. This module will then output these new frequency coefficients. We can test this module by feeding in some simple frequency mappings along with a fundamental frequency and observing the frequency coefficients that it outputs.

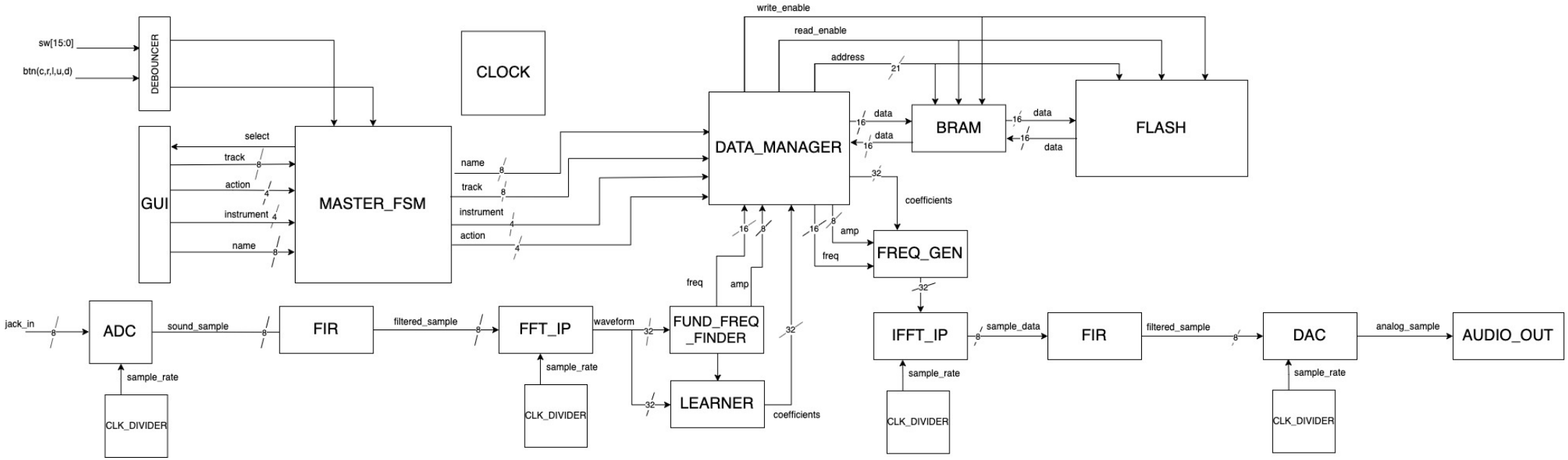
These modified frequency coefficients will then proceed to the IFFT IP module, which will take these coefficients and produce a waveform in the form of samples. We

can take as many samples as we need from each resulting waveform in order to meet our desired output frequency, which will probably be around 48 kHz. These samples will then be passed through an FIR filter in order to smooth them before we output them. The FIR filter should be relatively simple to add on here since we already implemented this in lab 5a. We can test the IFFT module by feeding in frequency coefficients that correspond to a relatively simple, known waveform and observing the output waveform as an analog wave.

Now that we have filtered waveform data, we need to convert the output back to analog. We can pass the output through a DAC in order to do so. This should be relatively simple to set up using Vivado's IP Catalog.

Data Management

Tyler will be the main contributor to this section of the project. In order to store music data, it will definitely be necessary to use a larger data storage than is available on the board. The plan is to use an SD card to store music and instrument data. Usually, writing to larger, nonvolatile memory is very slow, relatively. So, it is possible we may need to cache data in BRAM while we wait to read or write from the SD card. Either way, as you can see from the block diagram, we will need a data management unit for getting and setting data in its appropriate spot. Data will be sent into the unit along with control signals like track number, instrument name, and read or write from the FSM. Although we can hard code in addresses like the start of each track, there are variables, like the number of instruments, that will require us to store extra information, like a map of instrument names to memory addresses, which will require extra memory accesses in order to execute. It is also likely that we will have to read data from memory in blocks of some predetermined size, so we will have to process those blocks and we could possibly cache the extra data from them to increase the overall speed of our memory accesses. Since the specifics of reading and writing to memory are mostly internal to the data management unit, it will require some state to remember such things as how far along in reading a track it is. Another possible optimization we are thinking about is to store music data at staggered addresses (i.e. every 8th address) to be more efficient with playback.


sw[15:0]
btn(c,r,l,u,d)
DEBOUNCER
CLOCK
write_enable
read_enable
address
21
DATA_MANAGER
BRAM
FLASH
data
16
GUI
select
track
8
action
4
instrument
4
name
8
MASTER_FSM
name
8
track
8
instrument
4
action
4
32
coefficients
16
8
amp
freq
FREQ_GEN
32
jack_in
8
ADC
sound_sample
8
FIR
filtered_sample
8
FFT_IP
waveform
32
FUND_FREQ
_FINDER
32
IFFT_IP
sample_data
8
FIR
filtered_sample
8
DAC
analog_sample
AUDIO_OUT
sample_rate
CLK_DIVIDER
CLK_DIVIDER
32
LEARNER
coefficients
CLK_DIVIDER
CLK_DIVIDER